FGBS Spring 2022

Enabling Control-Flow Integrity with Pointer Authentication in FPGA SoC Platforms

Gabriele Serra*, Pietro Fara*, Giorgiomaria Cicero*
Francesco Restuccia†, Alessandro Biondi*

*Scuola Superiore Sant'Anna, Pisa
†University of California, San Diego

INTRODUCTION

Embedded systems

- OSes are written in C/C++
- Exposed to public access (especially in automotive and railway environments)

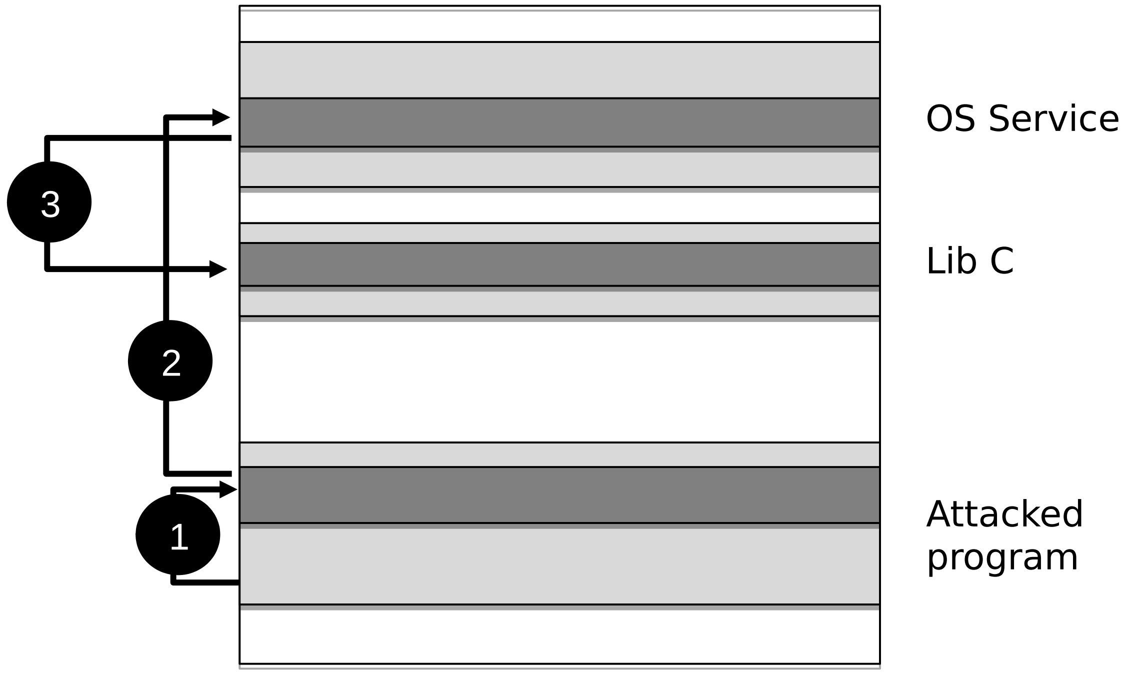
INTRODUCTION

Embedded systems

- OSes are written in C/C++
- Exposed to public access (especially in automotive and railway environments)

Susceptible to attacks

- **Code-Reuse-Attacks**
    - Re-use existent piece of code
    - I.e. flow deviated to gain root access
- Return-Oriented programming



INTRODUCTION

Embedded systems

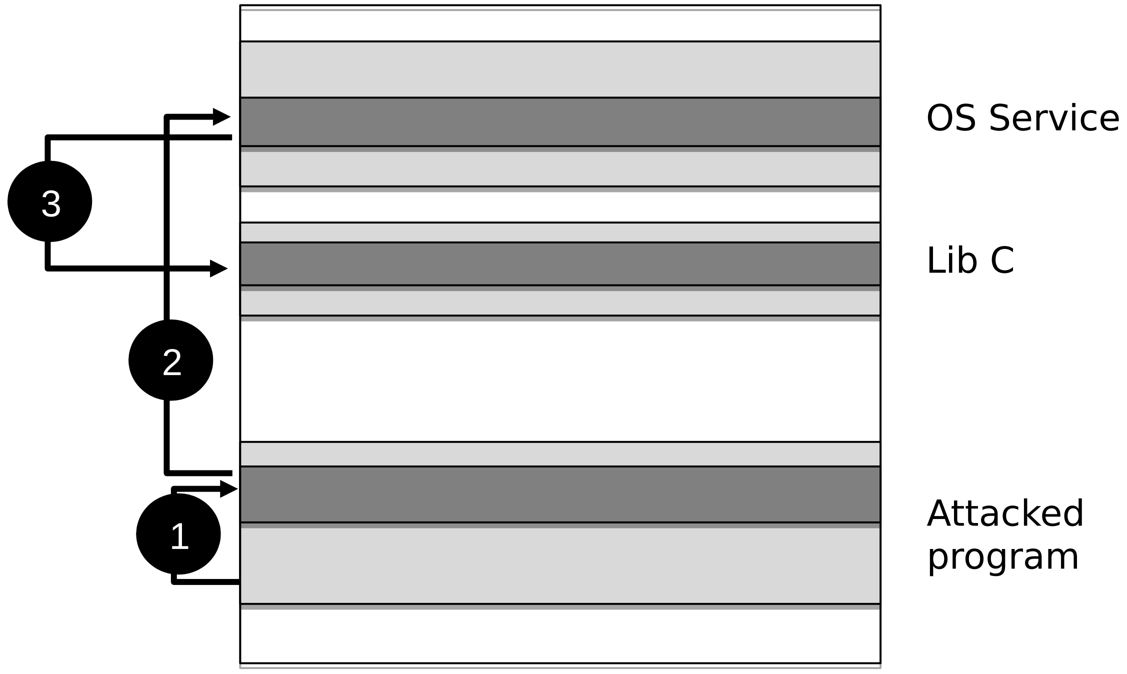
- OSes are written in C/C++
- Exposed to public access (especially in automotive and railway environments)

Susceptible to attacks

- **Code-Reuse-Attacks**
  - Re-use existent piece of code
  - I.e. flow deviated to gain root access
- Return-Oriented programming

Mitigation technique

- Address space layout randomization (ASLR)
- **Integrity check of control flow (CFI)**



INTRODUCTION

CFI basic idea:

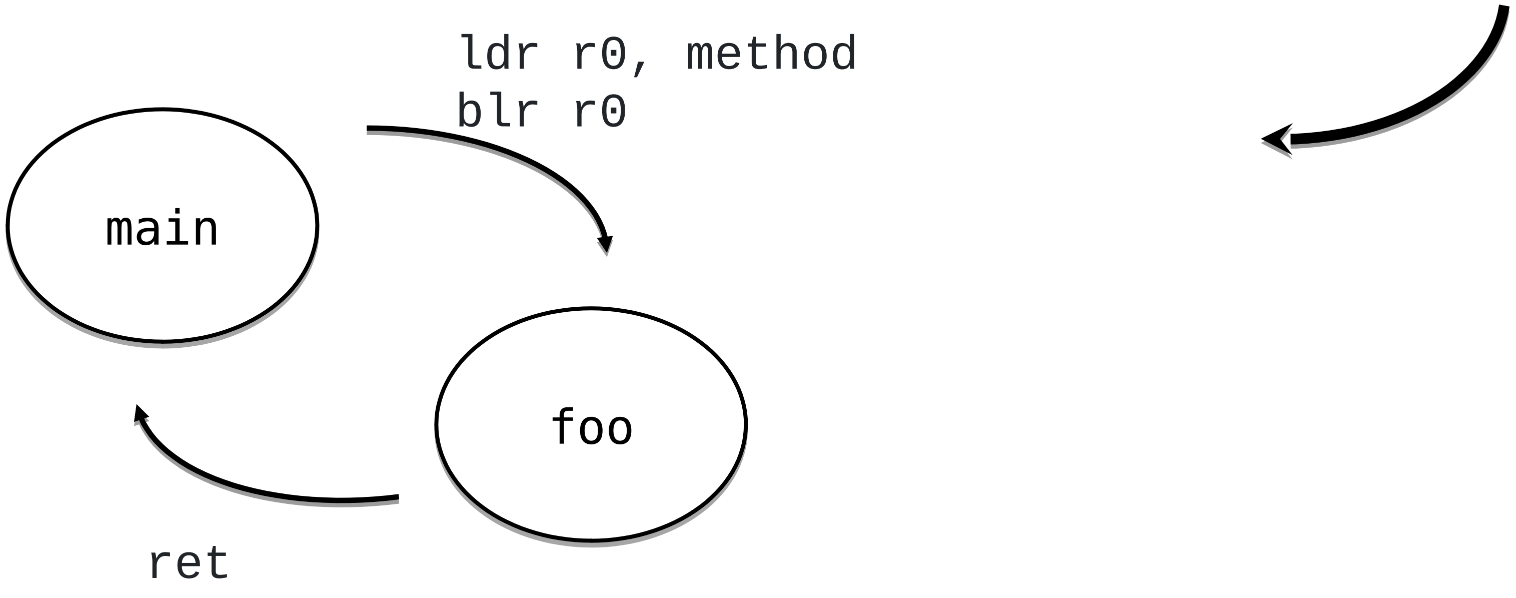
- build a Control Flow Graph (CFG) of the program
- **CFG** defines the legal execution

CFI basic idea:

- build a Control Flow Graph (CFG) of the program
- **CFG** defines the legal execution

```
void foo() { ... }

void main() {
    ...
    obj->method = foo;
    obj->method();
    ...
}
```



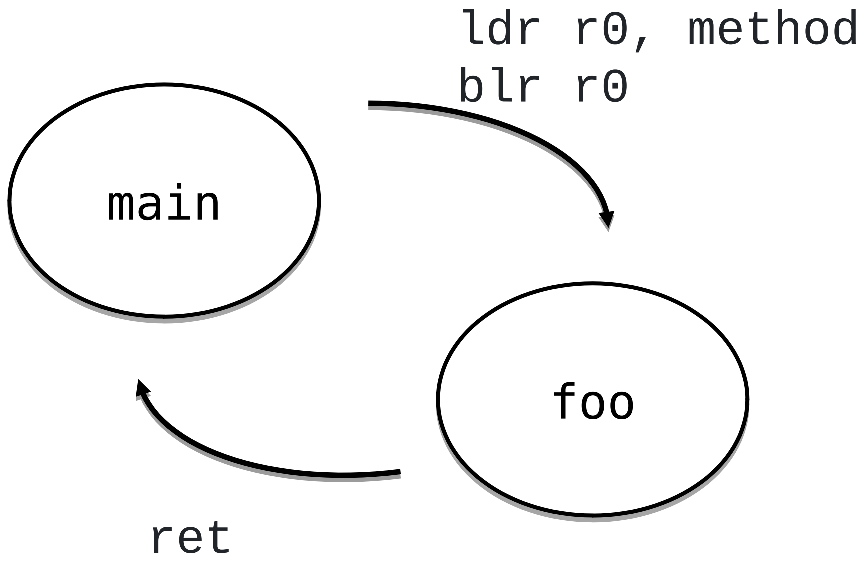
INTRODUCTION

CFI basic idea:

- build a Control Flow Graph (CFG) of the program
- **CFG** defines the legal execution

```
void foo() { ... }

void main() {
    ...
    obj->method = foo;
    obj->method();
    ...
}
```

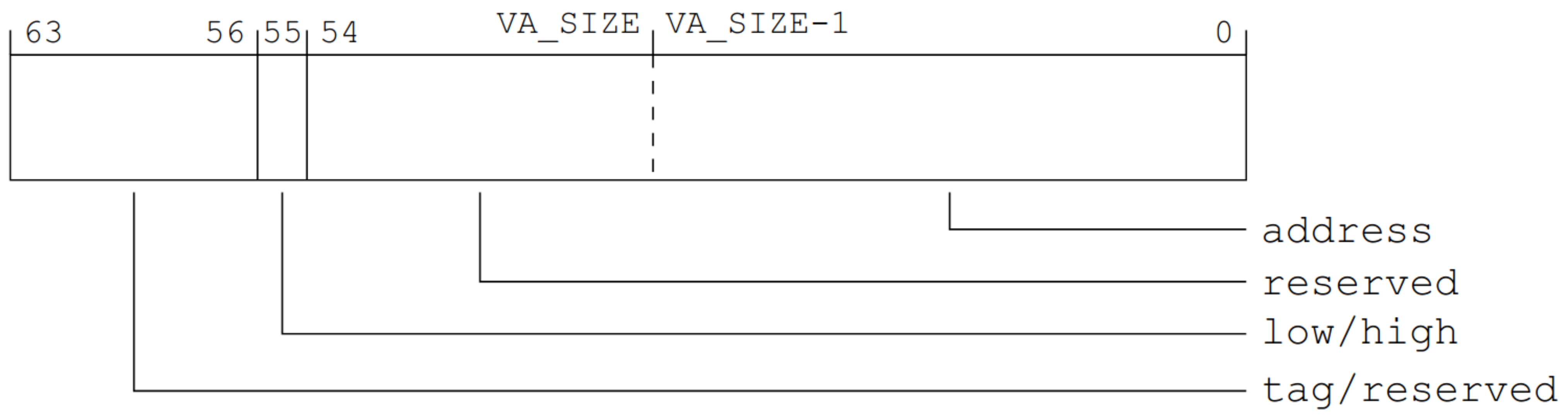


ARM introduced hw supports:

- Branch Targets Identification (BTI)
  - Forward branch protection
- **Pointer Authentication Code (PAC)**
  - Backward branch protection

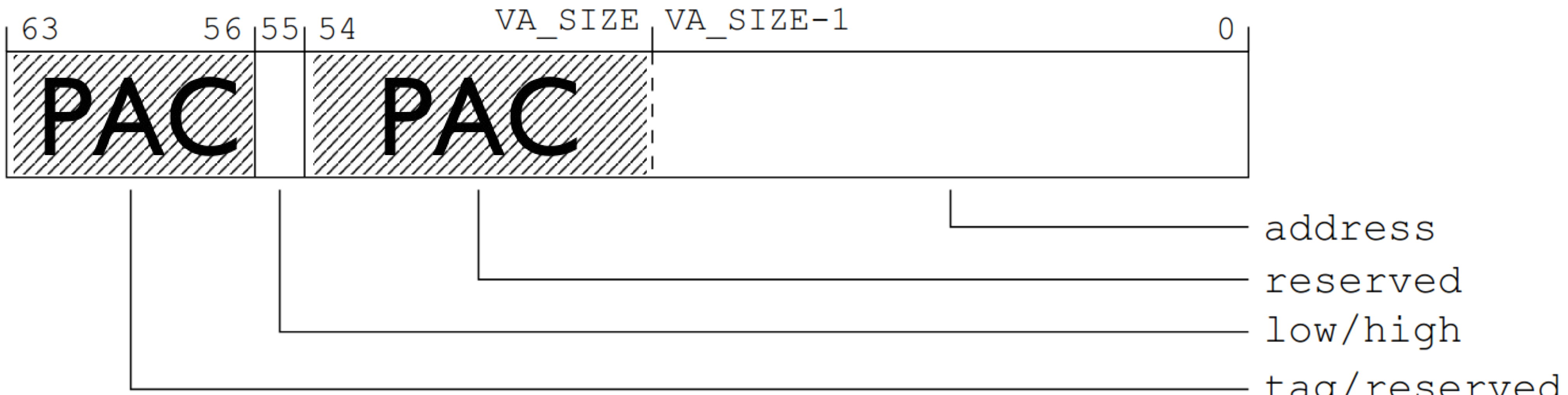
Pointers in AArch64:

- Address represented on [0:VA_SIZE]
- Typically VA_SIZE = 48
- Empty [VA_SIZE:54] and [56:63]



AArch64 Pointer Authentication Codes (PAC):

- Hardware-based CFI
- Leverages empty space on 64-bit virtual addresses
- Append a Message Authentication Code (MAC)

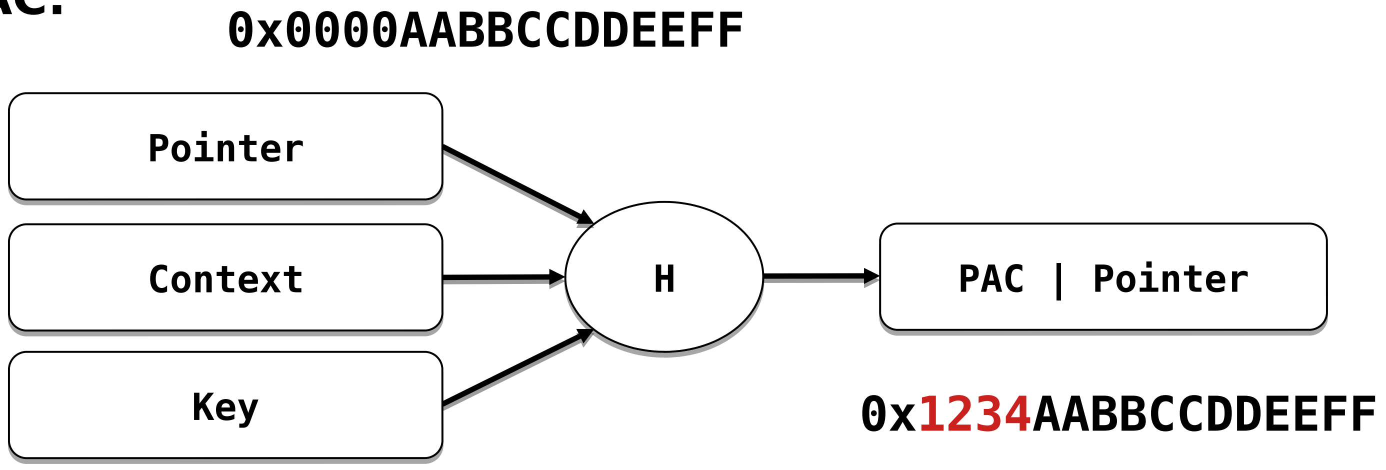


Introduced two insns:
- PAC
- AUTH

PAC Creation takes:
- A pointer
- A 64-bit context
- A 128-bit secret key

PAC algorithm ‘H’ can be:
- QARMA
- Implementation defined

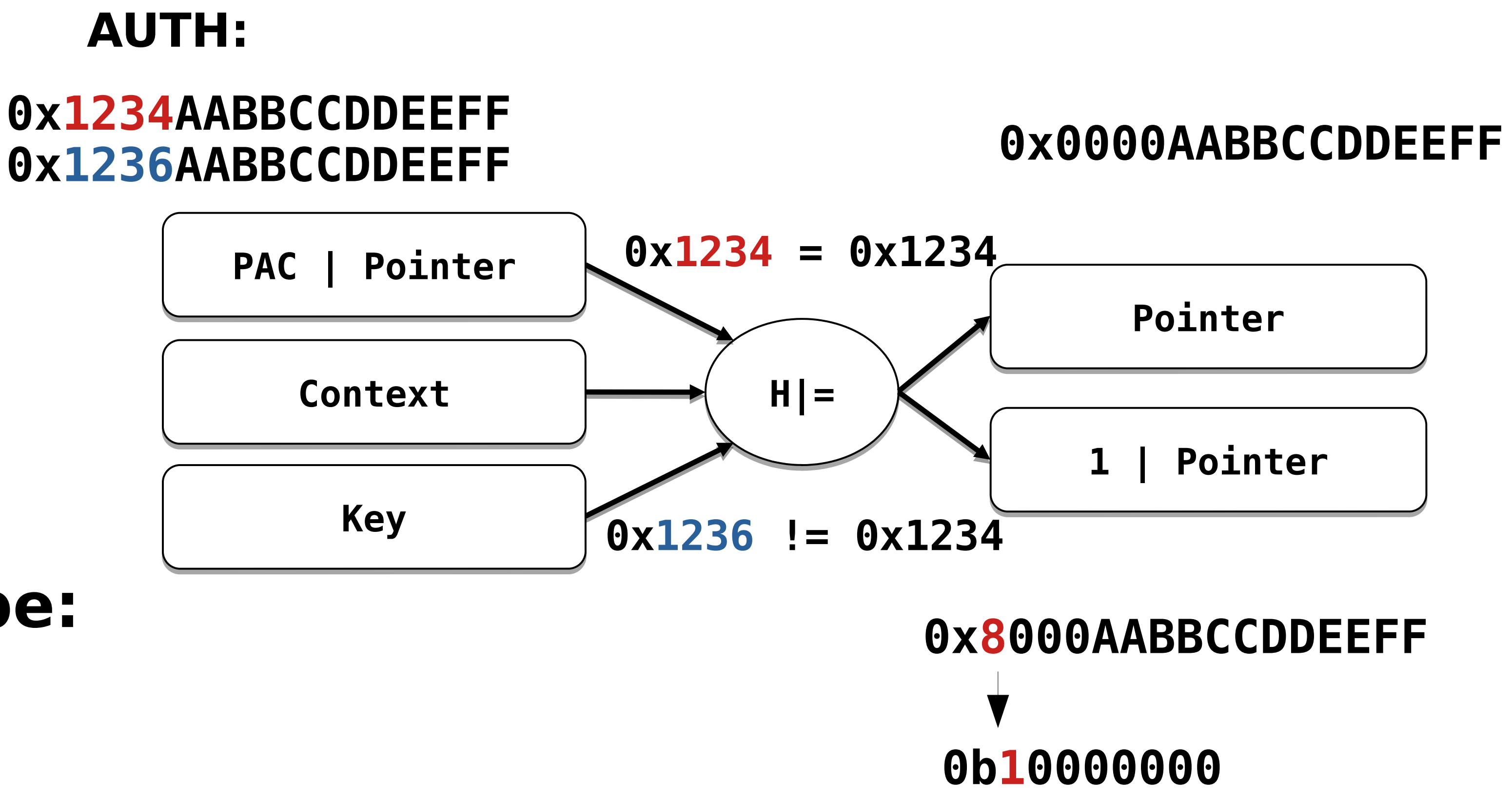
PAC:



Introduced two insns:
- PAC
- AUTH

PAC Creation takes:
- A pointer
- A 64-bit context
- A 128-bit secret key

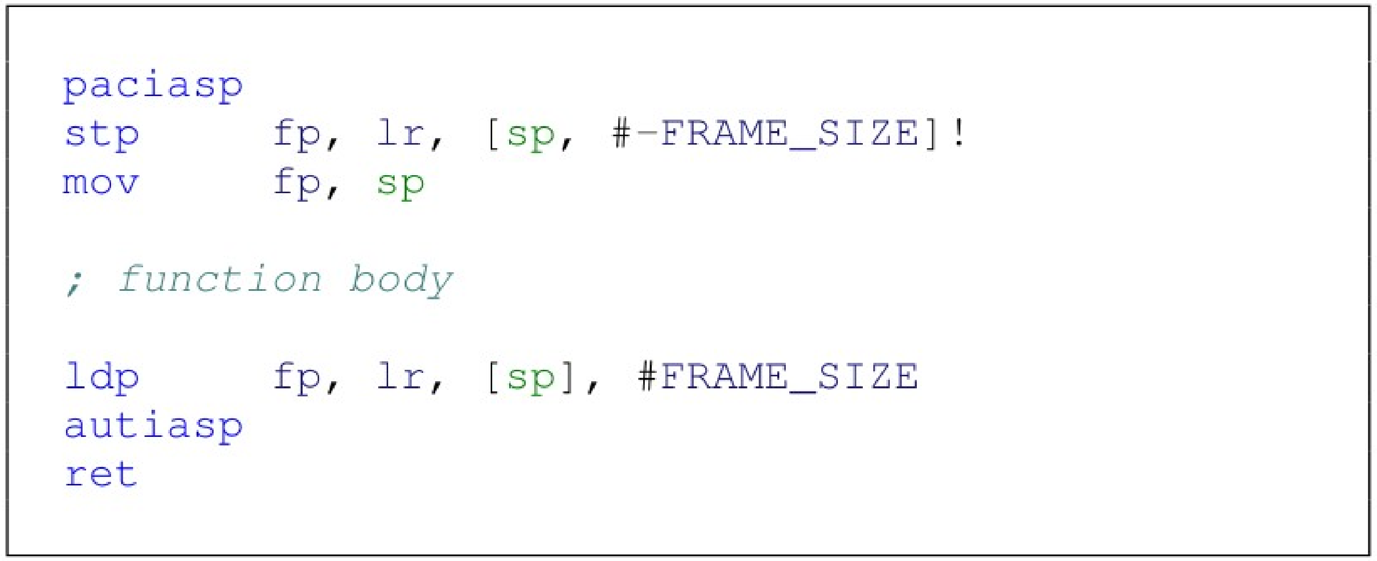
PAC algorithm 'H' can be:
- QARMA
- Implementation defined



Introduced two insns:
- PAC
- AUTH

PAC Creation takes:
- A pointer
- A 64-bit context
- A 128-bit secret key

PAC algorithm ‘H’ can be:
- QARMA
- Implementation defined



```
paciasp
stp     fp, lr, [sp, #-FRAME_SIZE]!
mov     fp, sp

; function body

ldp     fp, lr, [sp], #FRAME_SIZE
autiasp
ret
```

Pointer authentication ISSUES

- Weakness against signing gadget
- Weakness against kernel attackers
  - Cross EL/Key forgeries
  - Key memory leak
- Attack cannot be detected
  - Reported to ARM by Cicero et al in 2019
  - Will be fixed with FPAC in ARM v8.6
- **Available only on ARM ^v8.3**
- **Currently no COTS SoC available**

Pointer authentication ISSUES

- Weakness against signing gadget
- Weakness against kernel attackers
  - Cross EL/Key forgeries
  - Key memory leak
- Attack cannot be detected
  - Reported to ARM by Cicero et al in 2019
  - Will be fixed with FPAC in ARM v8.6
- **Available only on ARM ^v8.3**
- **Currently no COTS SoC available**

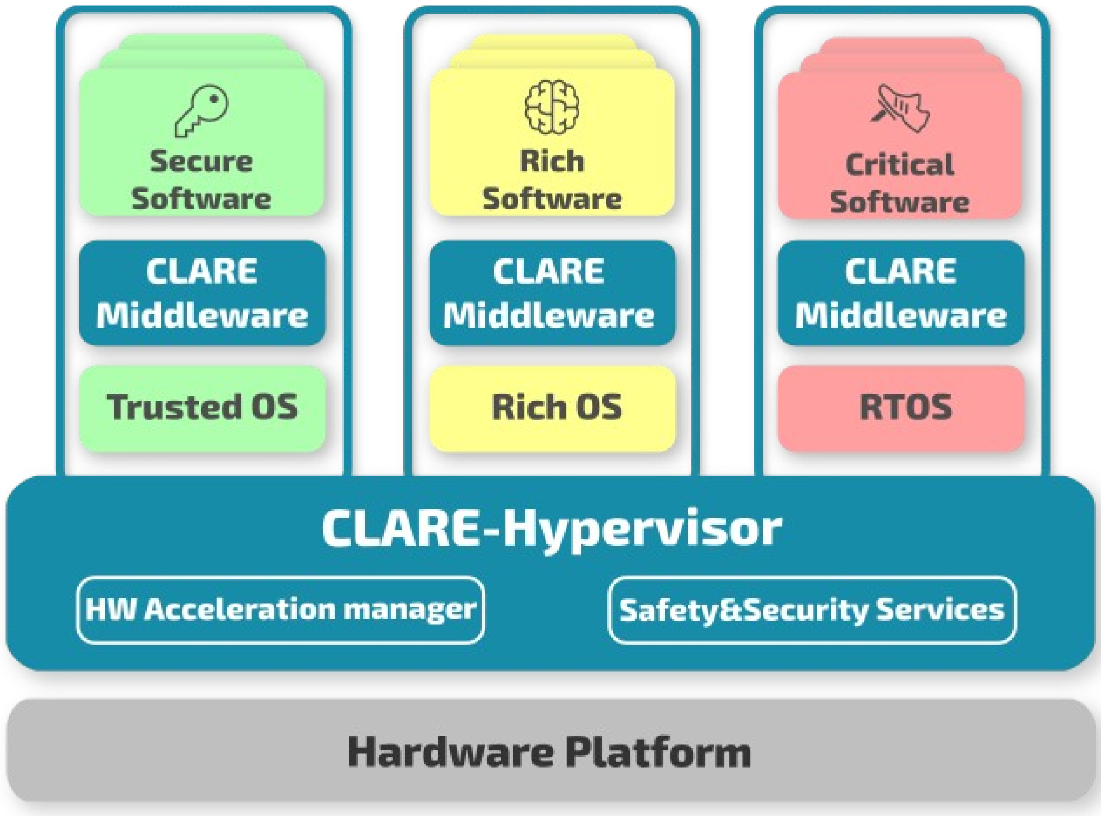
Leverage on PL & virtualization to counteract these issues!

CONTRIBUTIONS

CLARE

CLARE is a hypervisor-centric software stack. It simplifies the development cyber-physical systems offering:

- heterogeneous computing platforms support
- ready-to-use environment for deploying mixed-criticality applications.

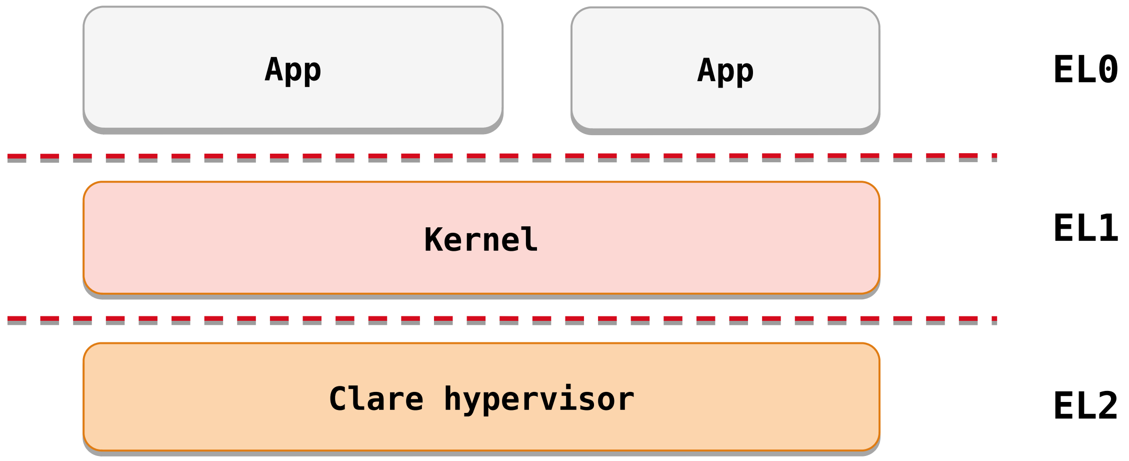


Check it out @ clare.santannapisa.it

CONTRIBUTIONS

Leverage on CLARE hypervisor to:

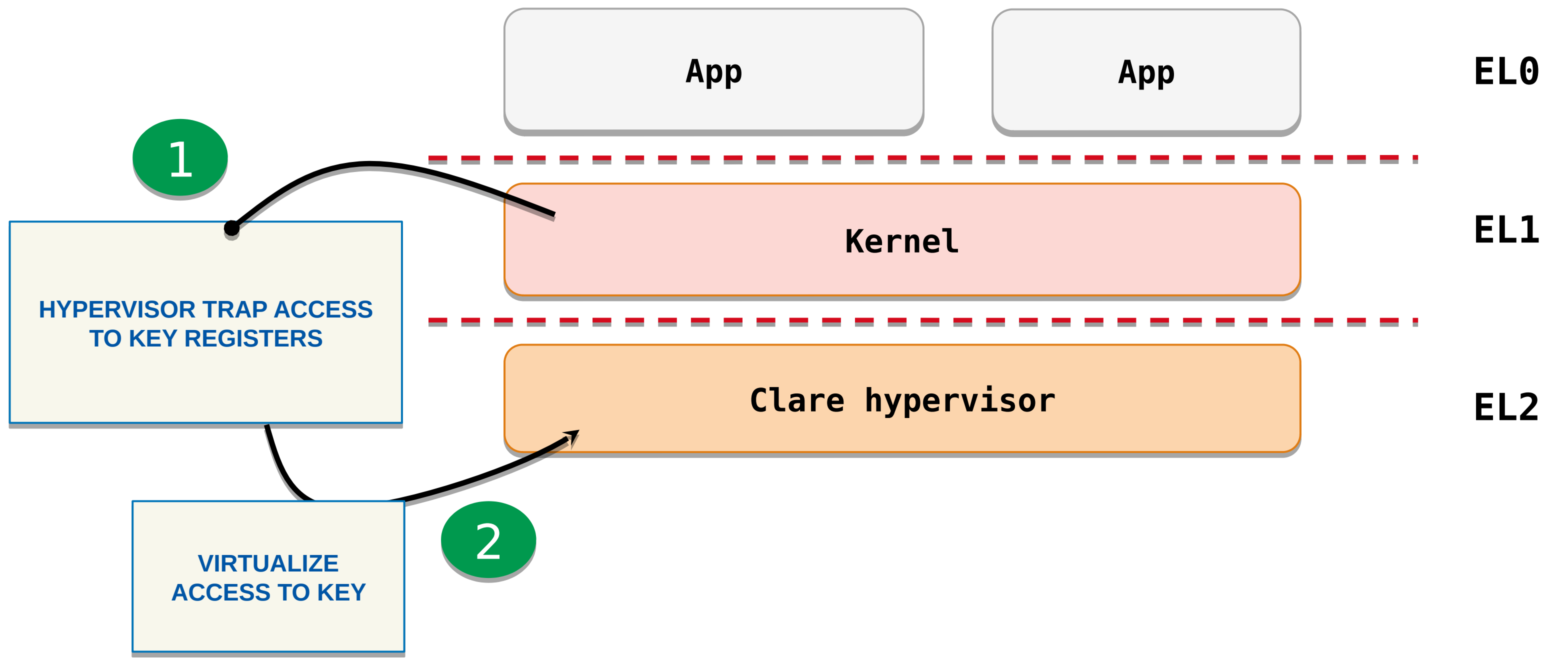
1. Improve key management
2. Provide PA to all AArch64 SoC



CONTRIBUTIONS

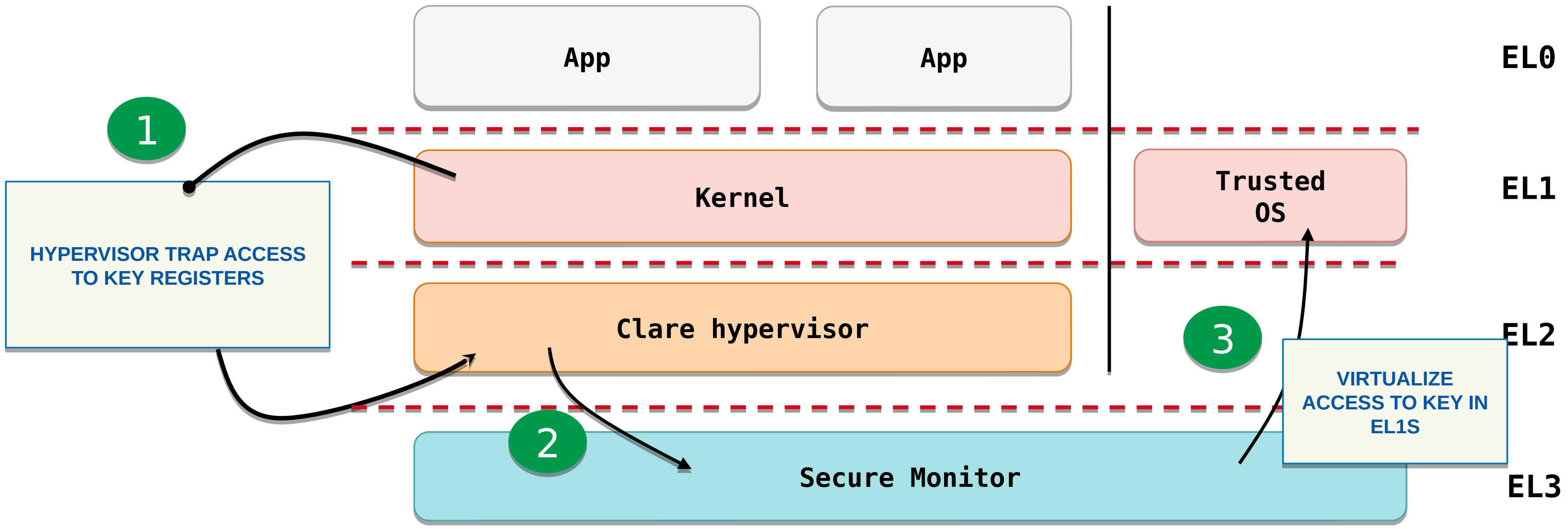
Leverage on CLARE hypervisor to:

1. Improve key management
2. Provide PA to all AArch64 SoC



Leverage on CLARE hypervisor to:

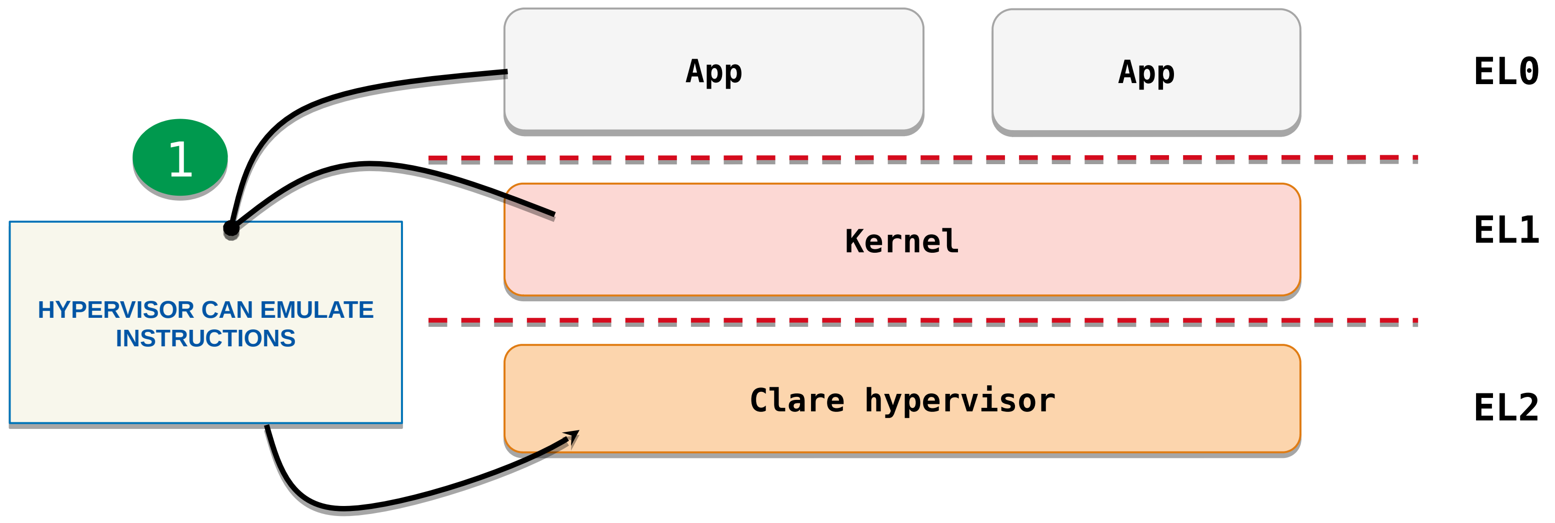
1. Improve key management
2. Provide PA to all AArch64 SoC


App
App
EL0
1
Kernel
Trusted OS
EL1
HYPERVISOR TRAP ACCESS TO KEY REGISTERS
Clare hypervisor
3
EL2
VIRTUALIZE ACCESS TO KEY IN EL1S
2
Secure Monitor
EL3

CONTRIBUTIONS

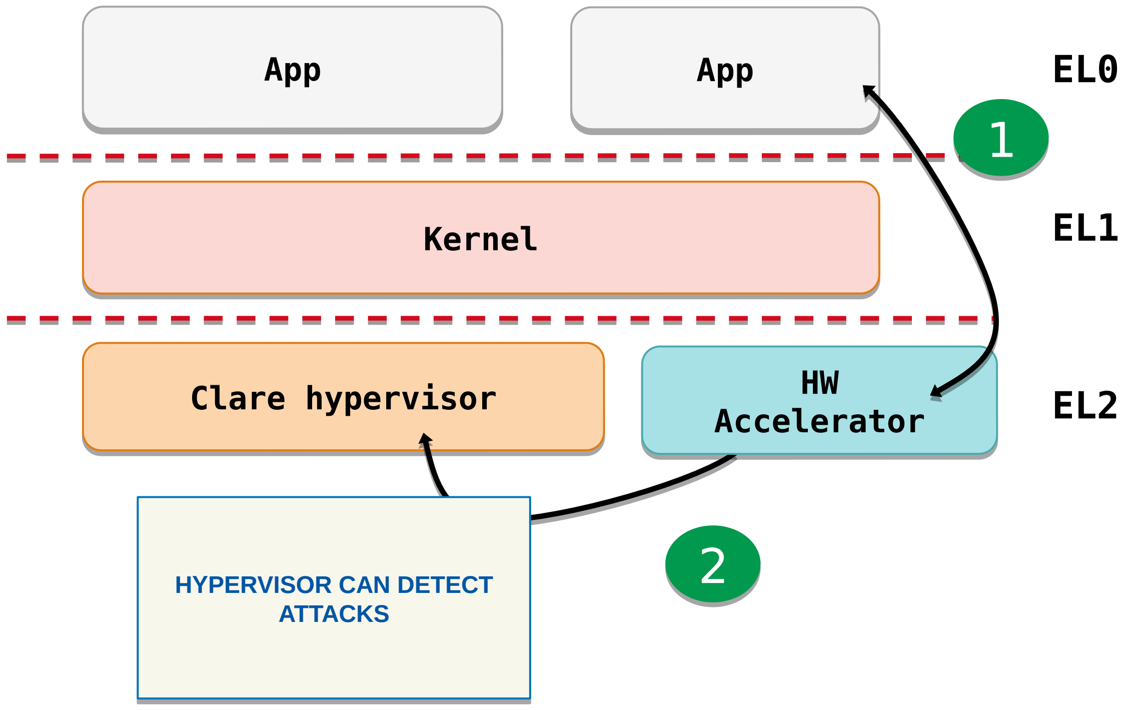
Leverage on CLARE hypervisor to:

1. Improve key management
2. Provide PA to all AArch64 SoC



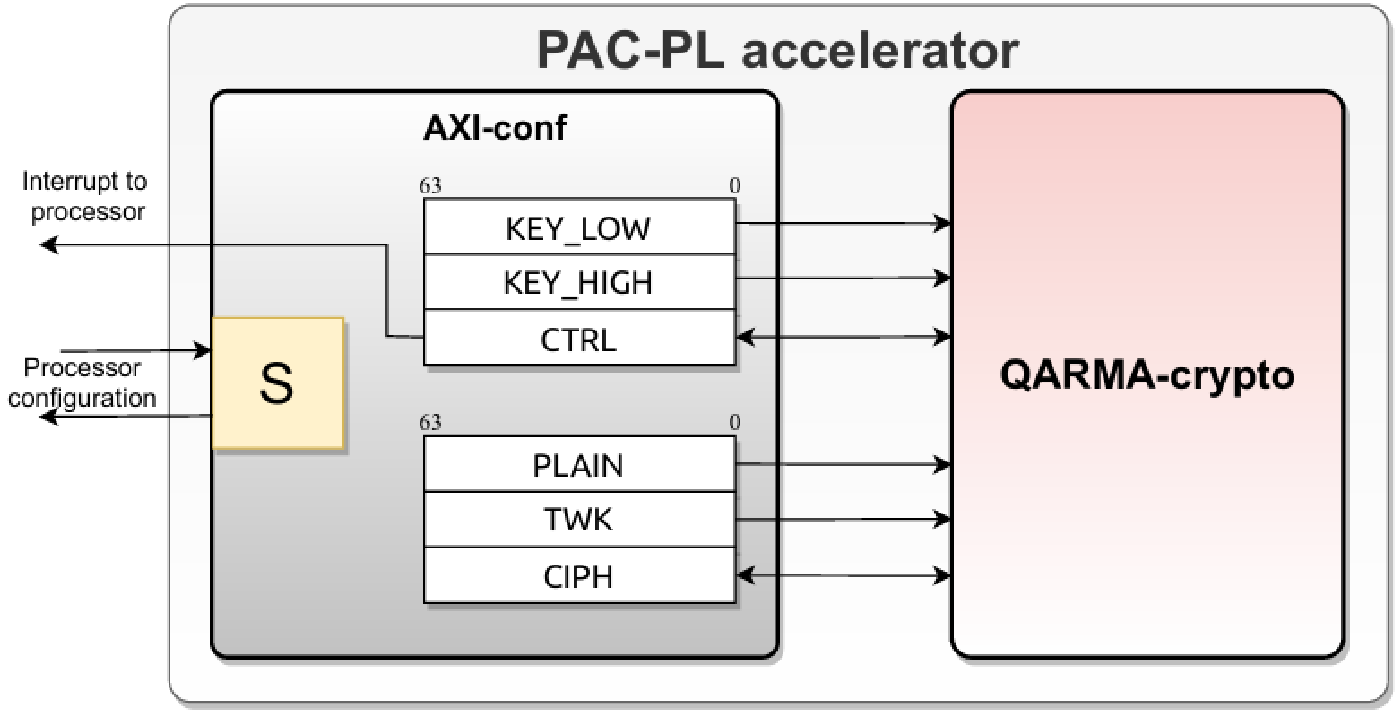
Leverage on CLARE hypervisor to:

1. Improve key management
2. Provide PA to all AArch64 SoC



Logic structure of PAC-PL HW accelerator

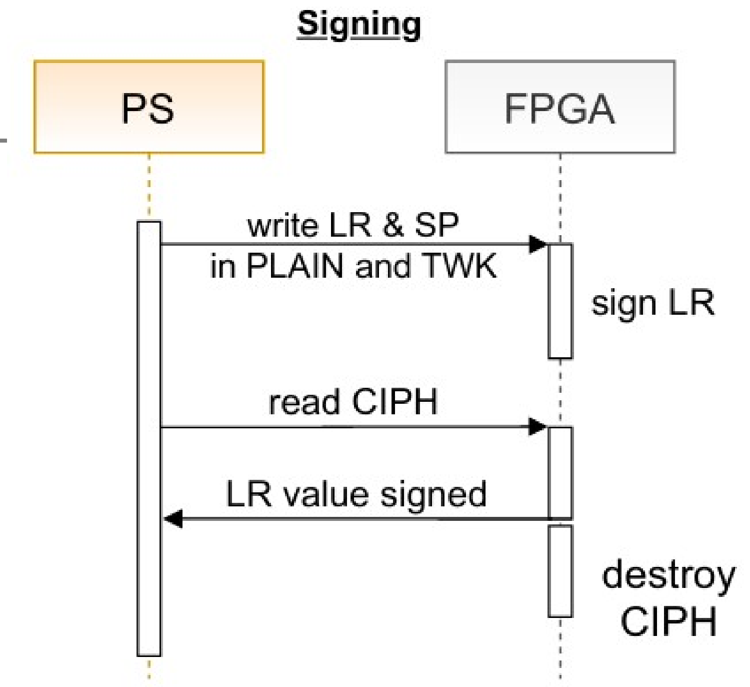
- Device registers are splitted in **two sets**, **privileged** and **non-privileged**.
- The device can send an **interrupt** to the CPU when **authentication fails**.



CONTRIBUTIONS

```
paciasp
stp      fp, lr, [sp, #-FRAME_SIZE]!
mov      fp, sp

; function body
```



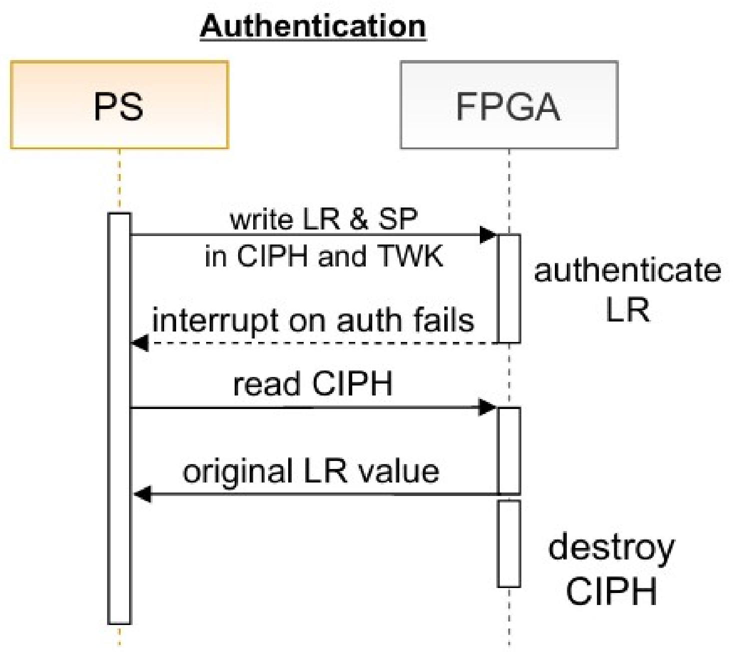
```
 mov x10, #DEV_BASE
 mov x9, sp
0:
 ; write lr and sp in the device
 stp lr, x9, [x10, #DEV_PLAIN_OFFSET]
 ; read the signed pointer from the device
 ldr x11, [x10, #DEV_CIPH_OFFSET]
 ; retry if the signed pointer is null
 cbz x11, 0b
 ; place the signed pointer in the link register
 mov lr, x11

 ; former prologue
 stp fp, lr, [sp, #-FRAME_SIZE]!
 mov fp, sp
```

CONTRIBUTIONS

```
; function body

ldp      fp, lr, [sp], #FRAME_SIZE
autiasp
ret
```



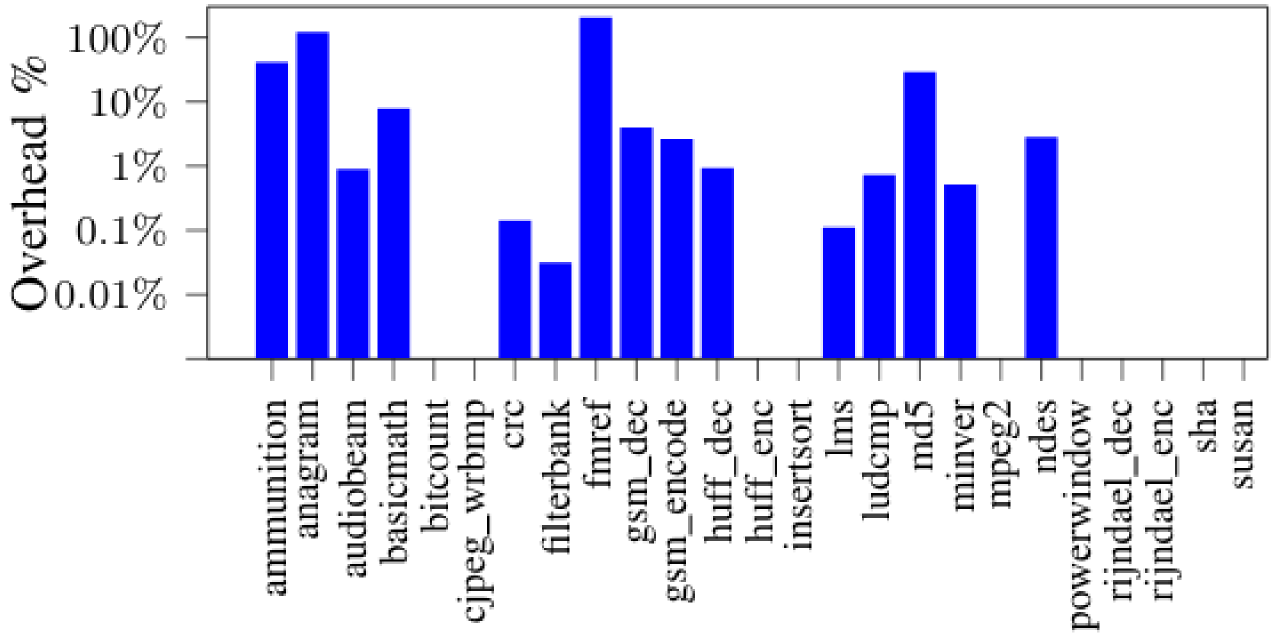
```
; former epilogue
ldp fp, lr, [sp], #FRAME_SIZE
; end of former epilogue

mov x10, #DEV_BASE
mov x9, sp
1:
; write sp and lr in the device
stp x9, lr, [x10, #DEV_TWK_OFFSET]
; read the authenticated pointer
ldr x11, [x10, #DEV_CIPH_OFFSET]
; retry if the authenticated pointer is null
cbz x11, 1b
; place the auth. pointer in the link register
mov lr, x11

ret
```

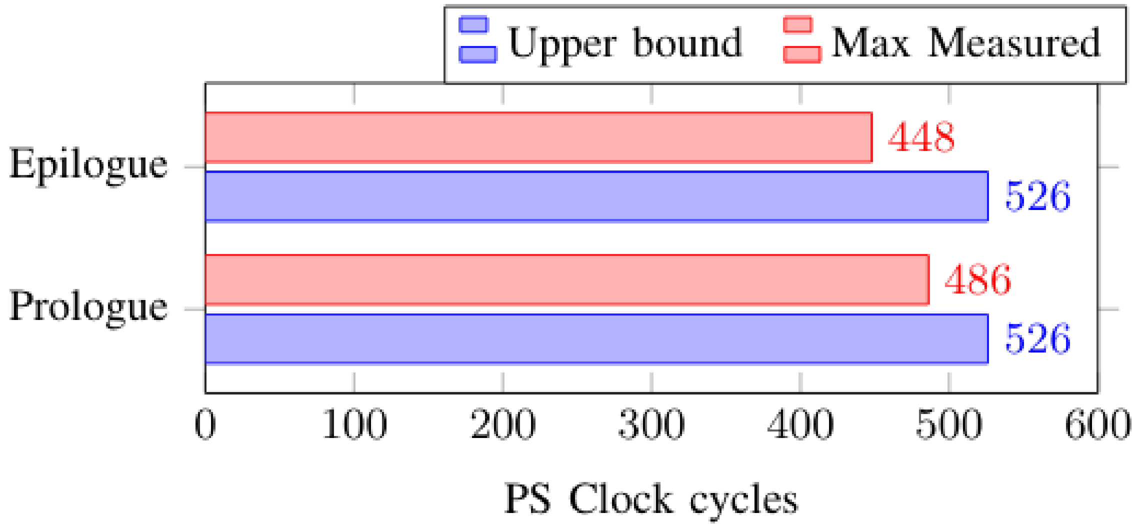
Overhead (%) for TACLeBench collection:

- In some benchmark the overhead was under timer resolution (µs)
- 21 out of 25 of them have **overhead below 10%** and the average overhead introduced is about 16.65%
- Each function protected by our plugin increases its footprint by 48 bytes.



“Analytic” and measured upper bounds:

- Hardware accelerator behavior was **measured** with a **System ILA**
- PS – PL write/read propagation **derived with a customer hardware device** probing 1000000 requests (w/ hot-caches on bare-metal firmware)



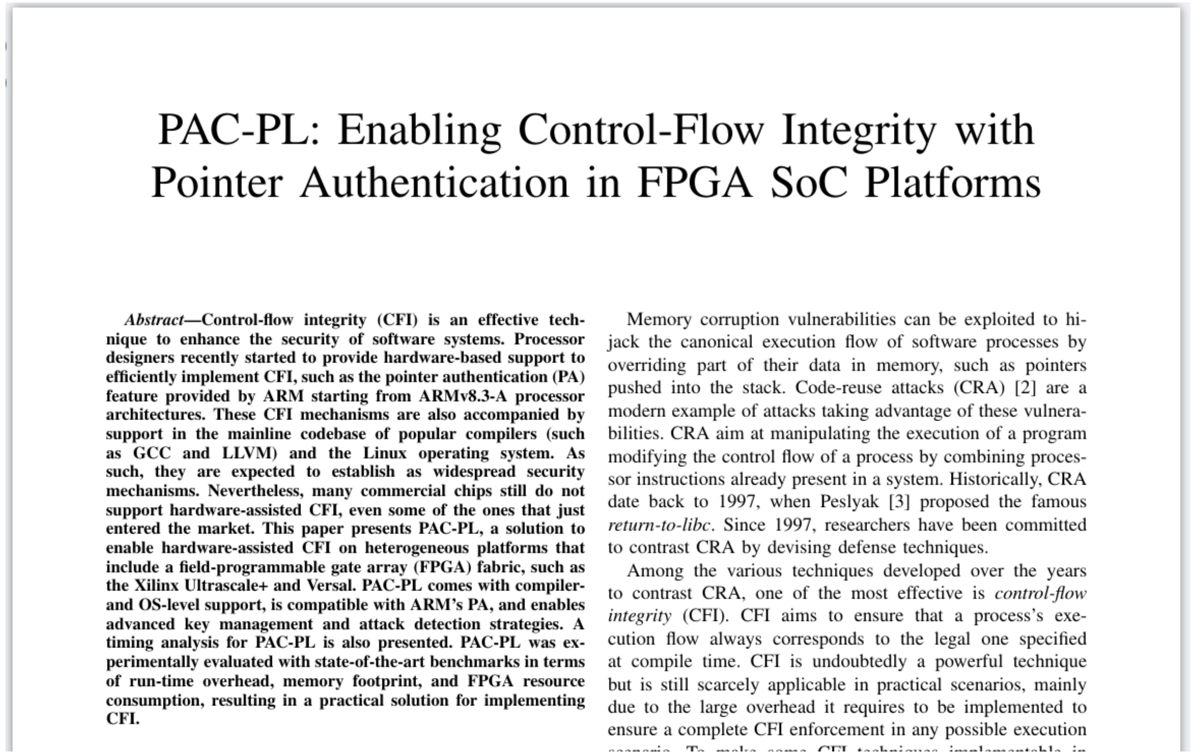
CONCLUSION & FUTURE DIRECTIONS

Issues

- Dumb (all-or-nothing) protection model
- The cost is heavy for recursive or call-intensive programs

Future directions

- Implement and test-out the same approach with a pure software emulation
- Make the protection model "smarter", analyzing the code and produce a specialized variant
- Tune up (at compile time) the protection degree based on the cost/vulnerability degree



PAC-PL: Enabling Control-Flow Integrity with Pointer Authentication in FPGA SoC Platforms

Abstract—**Control-flow integrity (CFI) is an effective technique to enhance the security of software systems. Processor designers recently started to provide hardware-based support to efficiently implement CFI, such as the pointer authentication (PA) feature provided by ARM starting from ARMv8.3-A processor architectures. These CFI mechanisms are also accompanied by support in the mainline codebase of popular compilers (such as GCC and LLVM) and the Linux operating system. As such, they are expected to establish as widespread security mechanisms. Nevertheless, many commercial chips still do not support hardware-assisted CFI, even some of the ones that just entered the market. This paper presents PAC-PL, a solution to enable hardware-assisted CFI on heterogeneous platforms that include a field-programmable gate array (FPGA) fabric, such as the Xilinx Ultrascale+ and Versal. PAC-PL comes with compiler- and OS-level support, is compatible with ARM's PA, and enables advanced key management and attack detection strategies. A timing analysis for PAC-PL is also presented. PAC-PL was experimentally evaluated with state-of-the-art benchmarks in terms of run-time overhead, memory footprint, and FPGA resource consumption, resulting in a practical solution for implementing CFI.**

Memory corruption vulnerabilities can be exploited to hijack the canonical execution flow of software processes by overriding part of their data in memory, such as pointers pushed into the stack. Code-reuse attacks (CRA) [2] are a modern example of attacks taking advantage of these vulnerabilities. CRA aim at manipulating the execution of a program modifying the control flow of a process by combining processor instructions already present in a system. Historically, CRA date back to 1997, when Peslyak [3] proposed the famous *return-to-libc*. Since 1997, researchers have been committed to contrast CRA by devising defense techniques.

Among the various techniques developed over the years to contrast CRA, one of the most effective is *control-flow integrity* (CFI). CFI aims to ensure that a process's execution flow always corresponds to the legal one specified at compile time. CFI is undoubtedly a powerful technique but is still scarcely applicable in practical scenarios, mainly due to the large overhead it requires to be implemented to ensure a complete CFI enforcement in any possible execution

Accepted @ RTAS (2022) - Milan (TBA)

THANK YOU. QUESTIONS?

Gabriele Serra
- gabriele.serra@santannapisa.it
- gabrieleserra.ml