Aalto University

UNIVERSITY OF WATERLOO

# Hardware-assisted Run-time Protection

*Thomas Nyman[‡], N. Asokan[†‡]*

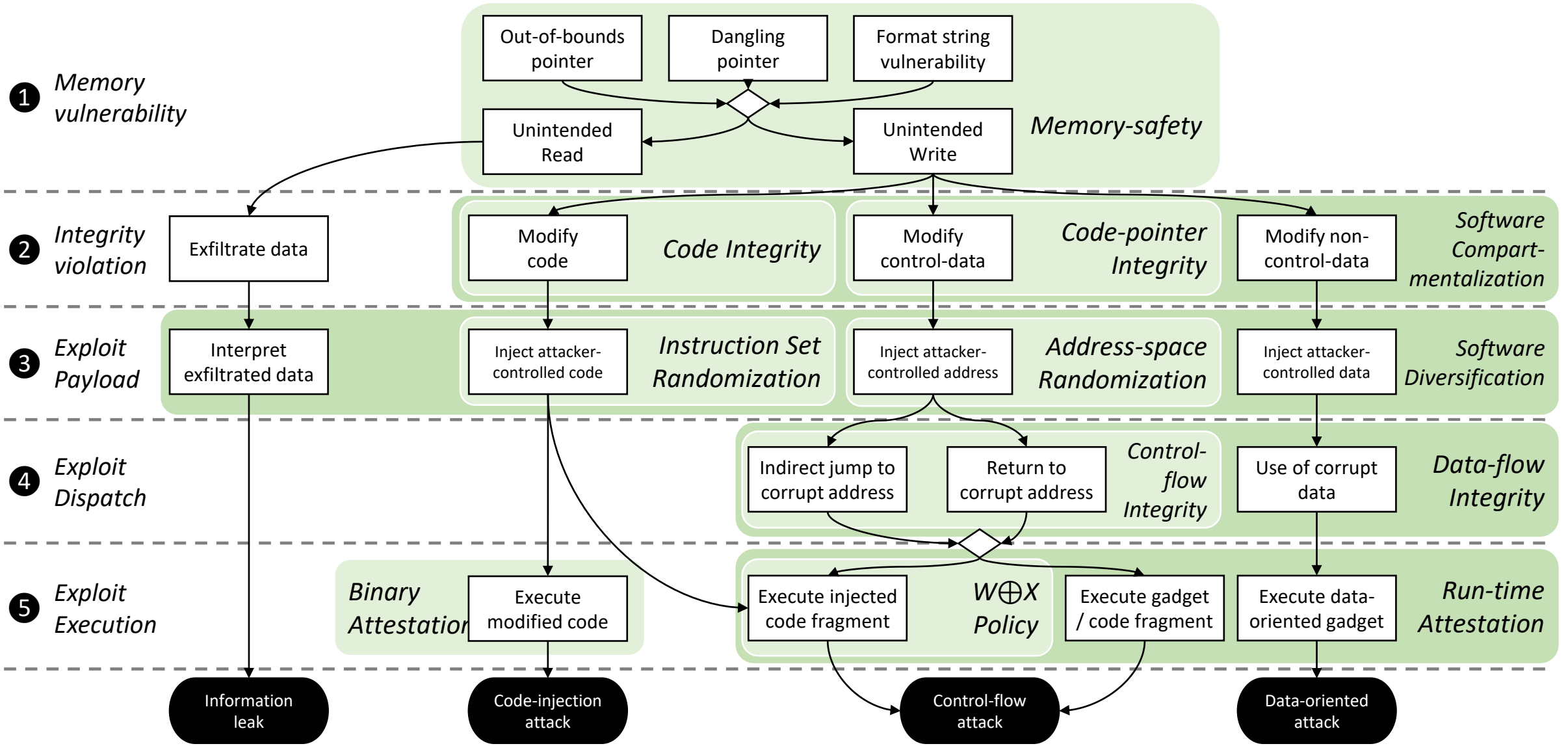
https://asokan.org/asokan/

@nasokan

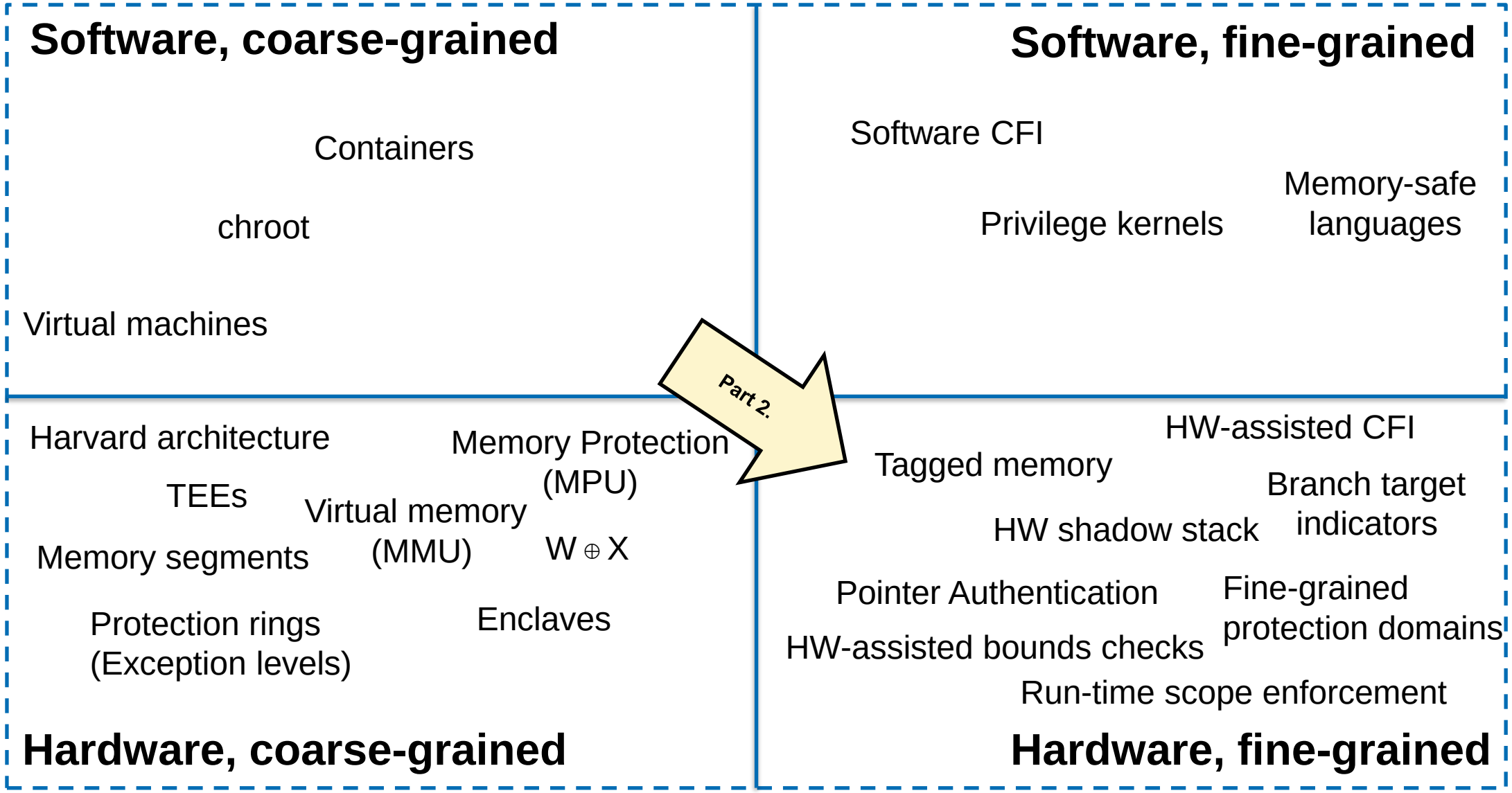
*Acknowledgements: Hans Liljestrand[†], Lachlan J Gunn[‡], Jan-Erik Ekberg[‡, §]*

*[†)] University of Waterloo, [‡)] Aalto University, [§)] Huawei Technologies*

# Taxonomy of Defenses




Adapted from Szekeres et al., SoK: Eternal War in Memory, IEEE SP (2013)


Software, coarse-grained
Containers
chroot
Virtual machines
Software, fine-grained
Software CFI
Memory-safe languages
Privilege kernels
Part 2.
Harvard architecture
TEEs
Memory segments
Protection rings (Exception levels)
Virtual memory (MMU)
Memory Protection (MPU)
W ⊕ X
Enclaves
Hardware, coarse-grained
HW-assisted CFI
Tagged memory
Branch target indicators
HW shadow stack
Pointer Authentication
Fine-grained protection domains
HW-assisted bounds checks
Run-time scope enforcement
Hardware, fine-grained

# Hardware-assisted defenses

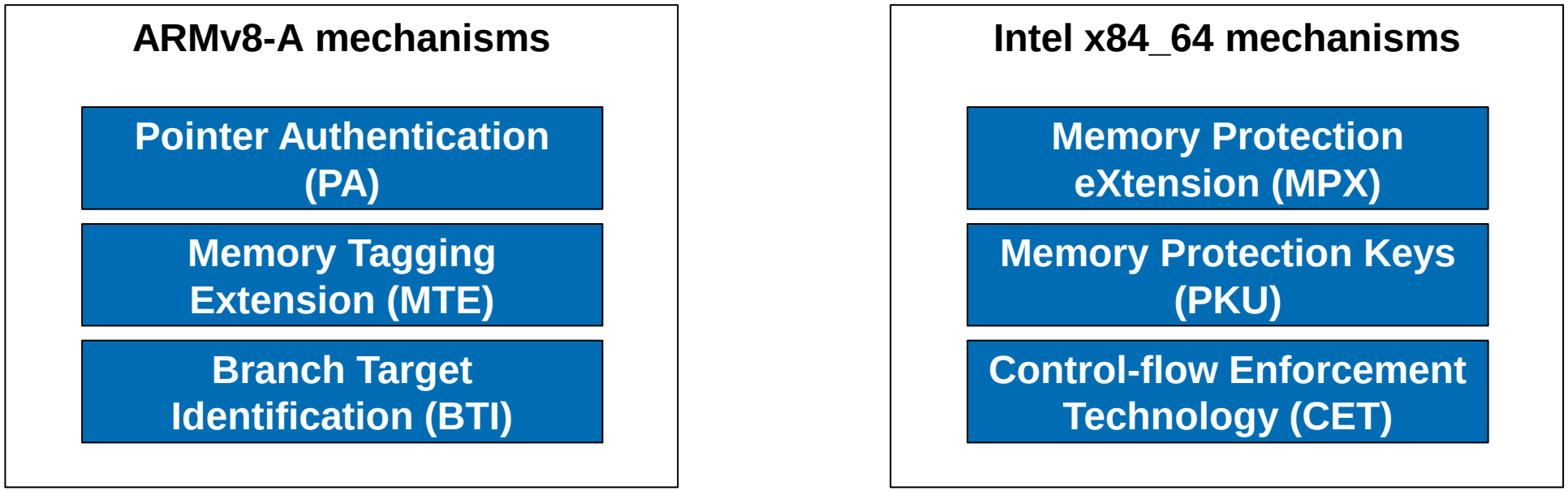
# How to thwart run-time attacks?

**Run-time attacks are now routine**

**Software defenses incur security vs. cost tradeoffs**

**Hardware-assisted defenses are attractive**

# Hardware assisted defenses in CotS processors

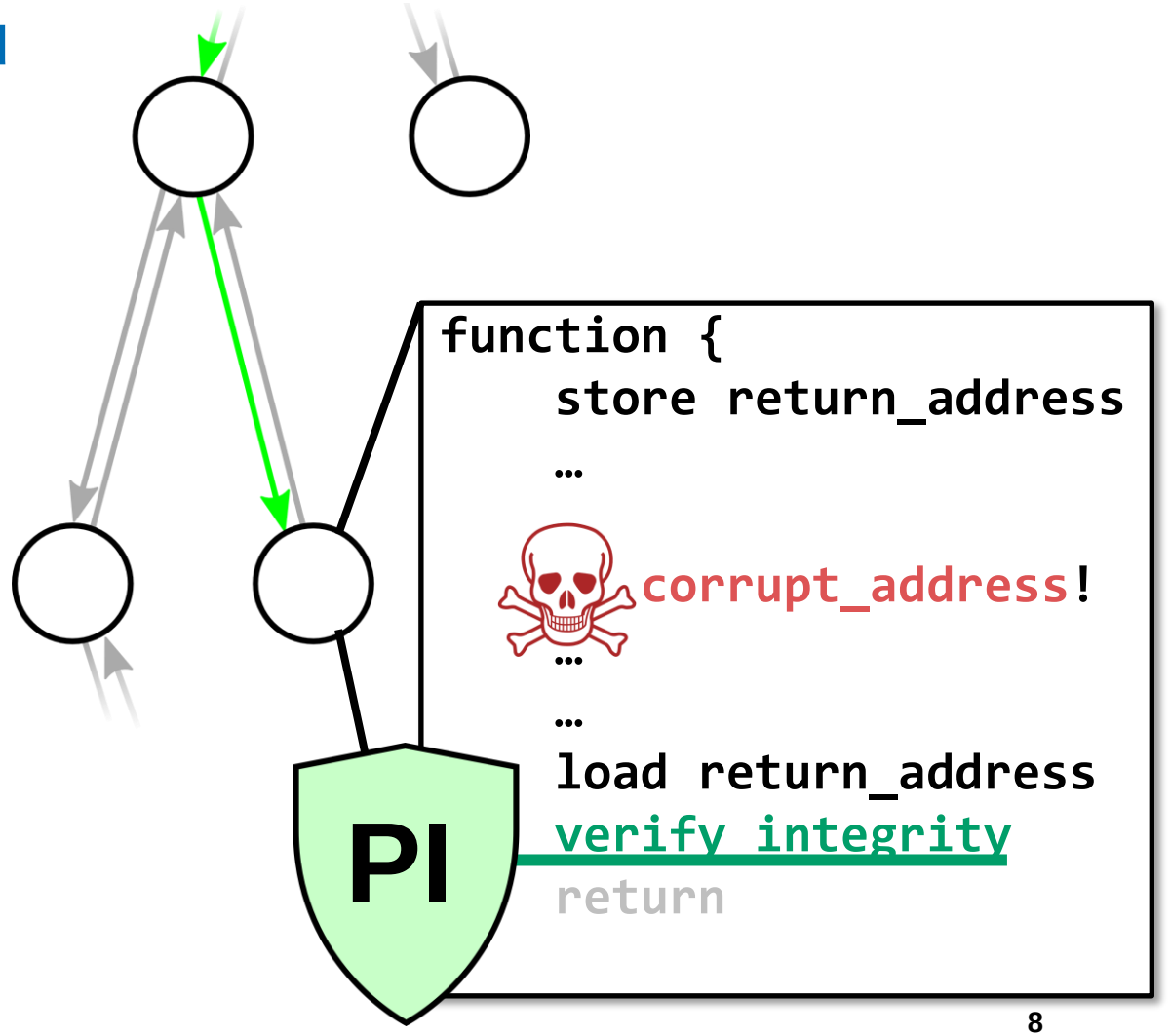


# ARMv8-A mechanisms

# Pointer Integrity: memory safety for pointers

**Ensure pointers in memory remain unchanged**

- **Code pointer integrity implies CFI**
  - Control-flow attacks manipulate code pointers
- **Data pointer integrity**
  - Reduces data-only attack surface




Kuznetsov et al. "Code-Pointer Integrity", USENIX OSDI 2014

# ARMv8.3-A Pointer Authentication


**General purpose hardware primitive approximating pointer integrity**

- Ensure pointers in memory remain unchanged

**Introduced in ARMv8.3-A specification** (2016) **to be improved in ARM-8.6-A** (2020)

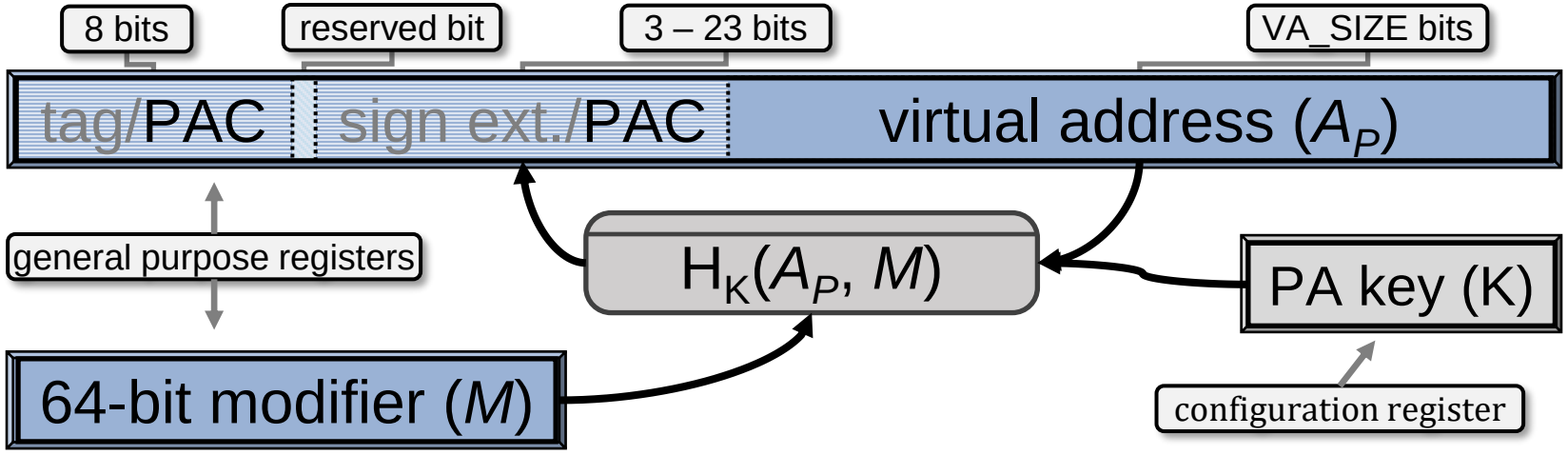
- First compatible processors 2018 (Apple A12 / iOS12)
- Support in Linux 5.0
- Instrumentation support in GCC 7.0 ( -msign-return address, deprecated in GCC 9.0 -mbranch-protection=pac-ret[+leaf] GCC 9.0 and newer)

ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019
ARM. Developments in the Arm A-Profile Architecture: Armv8.6-A. September 2019

# ARMv8.3-A PA – PAC Generation

**Adds Pointer Authentication Code (PAC) into unused bits of pointer**

- Keyed, tweakable MAC from pointer address and 64-bit modifier
- PA keys protected by hardware, modifier decided where pointer created and used




ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019

# ARMv8.3-A PA – Key management and instructions

## Keys for PAC generation and verification

| | |
|---|---|
| APIAKey_EL1 | Key A for instruction address PACs |
| APIBKey_EL1 | Key B for instruction address PACs |
| APDAKey_EL1 | Key A for data address PACs |
| APDBKey_EL1 | Key B for data address PACs |
| APGAKey_EL1 | Key for generic authentication |

## PA Instructions

| | |
|---|---|
| PAC<i\|d><a\|b> <Xd> <Xm> | Add PAC to address in *Xd* using modifier in *Xm* |
| AUT<i\|d><a\|b> <Xd> <Xm> | Authenticate address in *Xd* using modifier in *Xm* |
| PACGA <Xd> <Xn> <Xm> | Calculate generic PAC for data in *Xn* using modifier in *Xm* |
| XPAC<i\|d> <Xd> | Strip PAC for address in *Xd* |
| BRA<a\|b> <Xn> <Xm> | Branch to address in *Xn* after authenticating it with modifier in *Xm* |
| BLRA<a\|b> <Xn> <Xm> | As BRA but perform branch with link |
| RETA<a\|b> | Authenticate address in LR with SP as modifier and return |
| ERETA<a\|b> | Authenticate address in ELR with SP as modifier and exception return |
| LDRA<a\|b> <Xt> <Xn> | Authenticate address in *Xn* using modifier zero and load value to Xt |

operate on instruction keys only

operate on data keys only



ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019

# PA-based protection schemes

**PA instructions are primitives, assembled to form protection schemes**

**Two main components:**

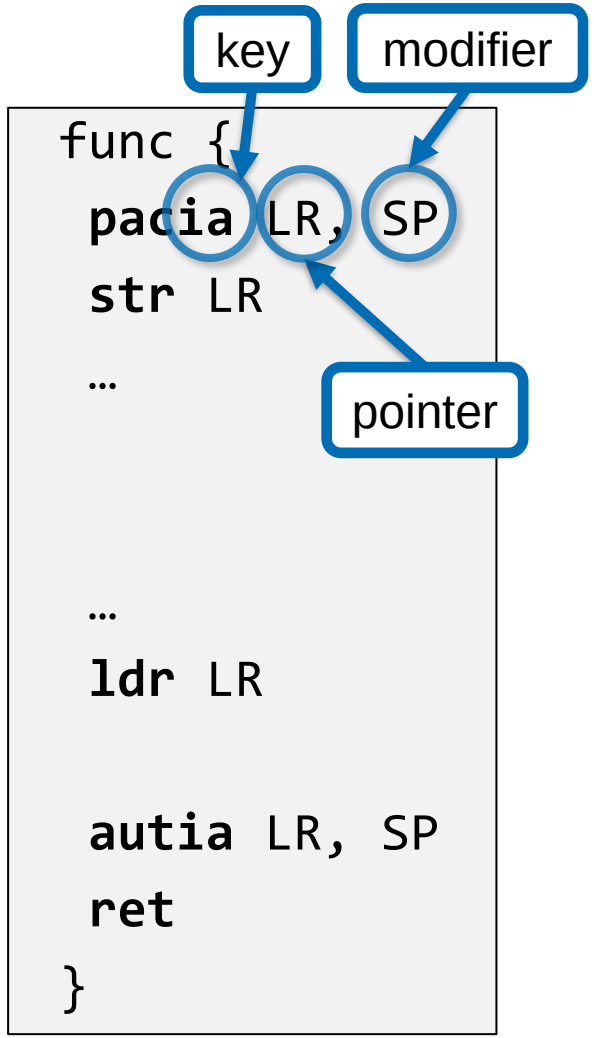
- When are pointers “PACed” and “unPACed”?
- Which modifier is used at a given point?

**What should the modifier be for a given pointer?**

- For security: using many different modifiers makes replay attacks harder
- For functionality: large numbers of modifiers are hard to keep track of

# PA prevents arbitrary pointer injection

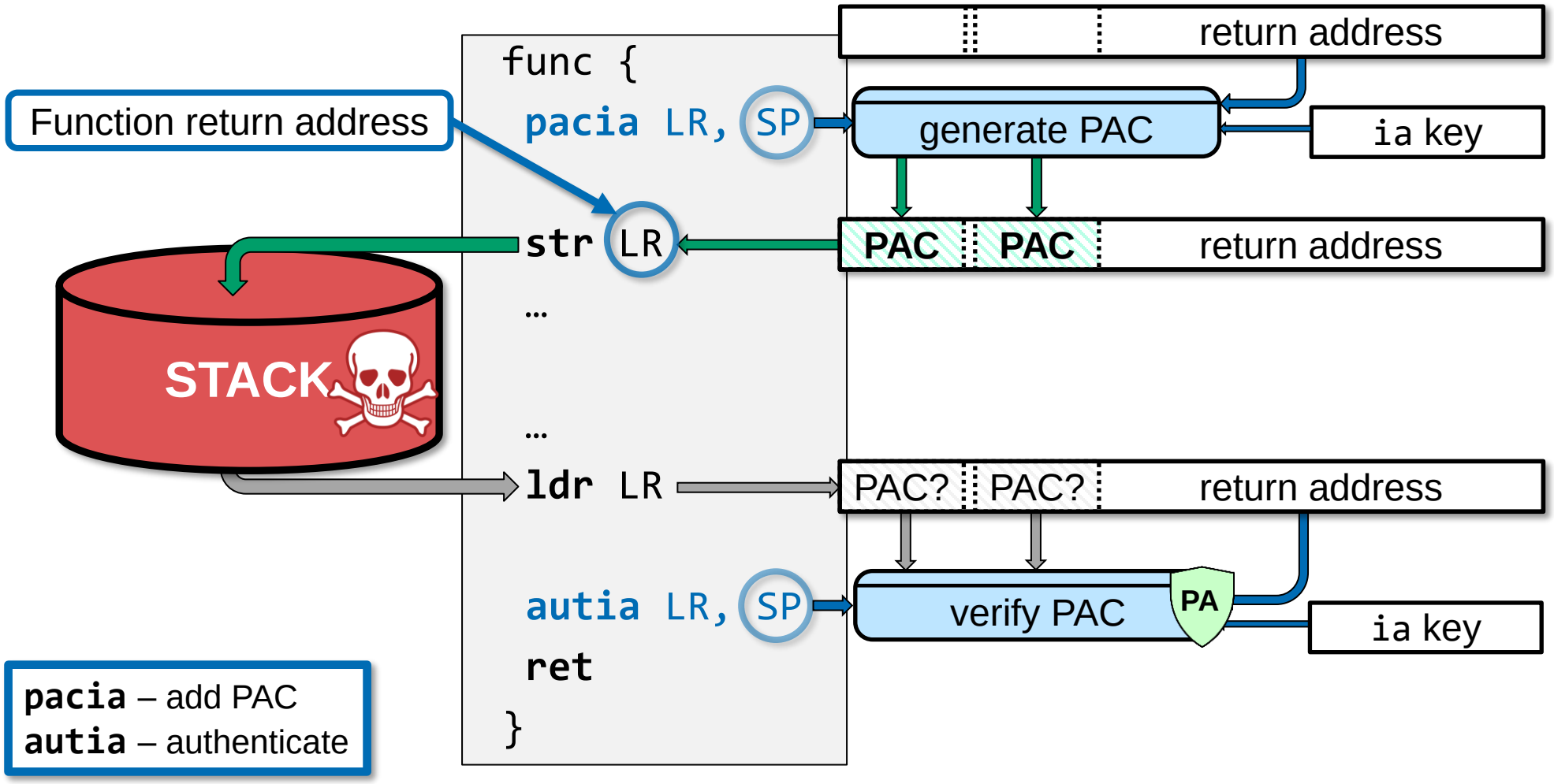
- **Modifiers do not need to be confidential**
  - Visible or inferable from the code section / binary
- **Keys are protected by hardware and set by kernel**
  - Attacker cannot generate PACs




**pacia** – add PAC
**autia** – authenticate

# Example: `-msign-return-address`

**Deployed in GCC 5.0 and LLVM/Clang 7.0**




Qualcomm "Pointer Authentication on ARMv8.3", whitepaper 2017

# PA return address protection as a canary

**The signed return address effectively is a canary:**

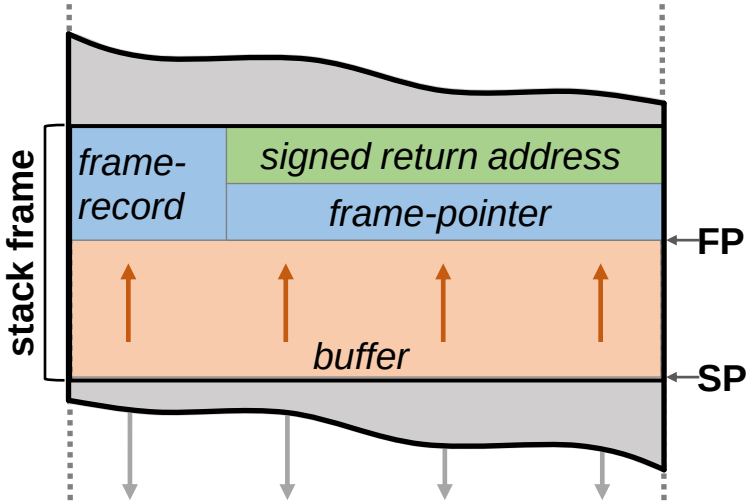
- Any overflow that corrupts the return address is detected

**More powerful than `-stack-protector` canaries:**

- Does not require reference value
- Can be bound to contextual information (e.g., the SP value)
- Protects return address against arbitrary writes

**Also has similar weaknesses:**

- Existing return addresses can be reused

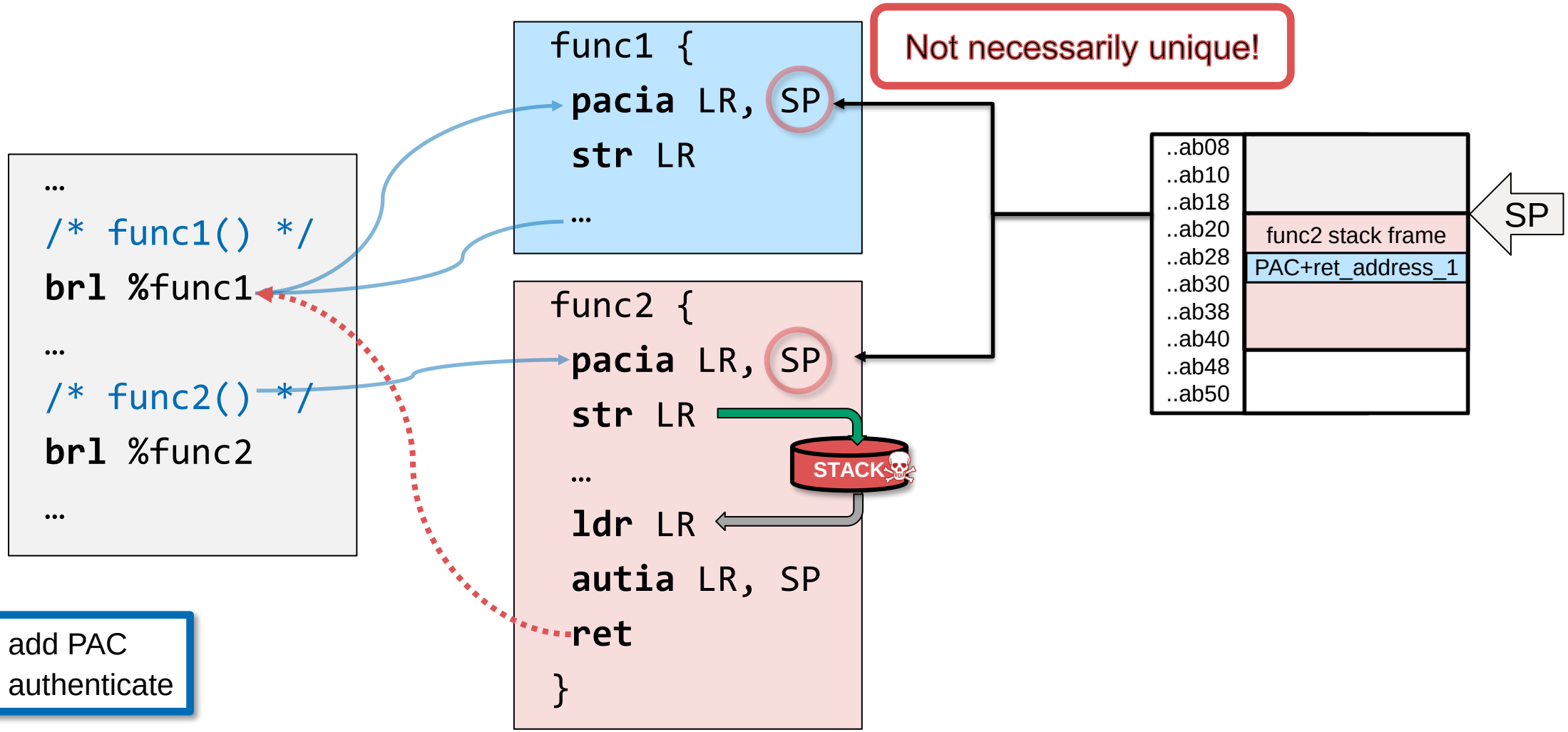





Liljestrand et al. PAC it up: Towards Pointer Integrity using ARM Pointer Authentication USENIX Security (2019)

# PA only approximates fully-precise pointer integrity

## Adversary may reuse PACs




**pacia** – add PAC
**autia** – authenticate

# PARTS

**Modifier: based on pointer type**

- Assigned at compile-time based on C type
- “this pointer really points to this type of data or function”

**On-load, on-branch authentication**

- Branching with combined auth+branch instruction (**`lbraa`**)
- Iterating an array uses only one authentication

```
// ptr = …
  …
  mov Xmod, #type_id
  pacia Xptr, Xmod
```

PACed only on pointer creation!

```
// *ptr
…
ldr Xptr, <memory>
mov Xmod, #type_id
autda Xptr, Xmod
<something> [Xptr]
```

```
// ptr();
  …
  …
  mov Xmod, #type_id
  lbraa Xptr, Xmod
  …
```

Authenticated on use

| | |
|---|---|
| **`pacda`** | – add PAC with data A-key |
| **`autda`** | – authenticate |
| **`pacia`** | – add PAC with instr A-key |
| **`lbraa`** | – authenticate and branch |

Liljestrand et al. PAC it up: Towards Pointer Integrity using ARM Pointer Authentication USENIX Security (2019)

# ARMv8.5-A Memory Tagging Extension


**Ensures memory accesses are safe by comparing tag in pointer with tag in memory**

**Introduced in ARMv8.5-A specification** (announced September 2018)

- Support in Linux proposed July 2019
- Stack Tagging will become available in LLVM 9
- Heap Tagging support planned

ARM. Armv8.5-A Memory Tagging Extension, whitepaper 2019
ARM. Opensource support for Armv8.5-A Memory Tagging Extension. 2019

# ARMv8.5-A MTE

**Address tags stored in top byte of pointer**

- uses existing top-byte ignore feature

**Allocation tags stored by transparently by hardware and cached**

- 4-bit tag per 16-byte granule of memory

**Mismatch between tags reported either:**

- synchronously (precise check during testing), or
- asynchronously (imprecise checks after deployment)

ARM. Armv8.5-A Memory Tagging Extension, whitepaper 2019

# ARMv8.5-A MTE

char *ptr = new char [16]; // memory colored

ptr[17] = 42; // color mismatch -> overflow

delete [] ptr; // memory re-coloured on free

ptr[10] = 10; // color mismatch -> use-after-free



https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/enhancing-memory-safety

# ARMv8.5-A MTE – Instructions

**MTE Instructions**

| | |
|---|---|
| `IRG` | Insert random address tag to address in register |
| `GMI` | Manipulate excluded set of tags for `IRG` |
| `ADDG/SUBG` | Arithmetic on addresses with tags for creating pointer to objects on stack |
| `SUBP(S)` | 56-bit subtract allowing address tag in top byte of pointers to be ignored |
| `LDG/STG/STZG` | Get or set allocation tags for granule (`STGZ` also initializes data to zero) |
| `ST2G/STZ2G` | Like `STG` or `STZG` but operate on two granules of memory at a time |
| `STGP` | Store both tag and data to memory |
| `LDGM/STGM/STZGM` | Bulk tag manipulation for initializing or serializing tags by system software |

ARM. Armv8.5-A Memory Tagging Extension, whitepaper 2019
ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019

# ARMv8.5-A Branch Target Identification


**Hardware-assisted CFI similar to Intel CET Indirect Branch Tracking**

**Introduced in ARMv8.3-A specification** (2016)

- Support for Linux proposed May 2019
- Instrumentation support in GCC 9.0 (-mbranch-protection=standard|bti)

ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019

# ARMv8.5-A BTI

**Indirect branches to guarded code regions require marker instructions**

- compiler places marker potential indirect branch targets
- two classes of targets: calls and jumps (`RET` instructions not restricted by BTI)

**Branch sources**

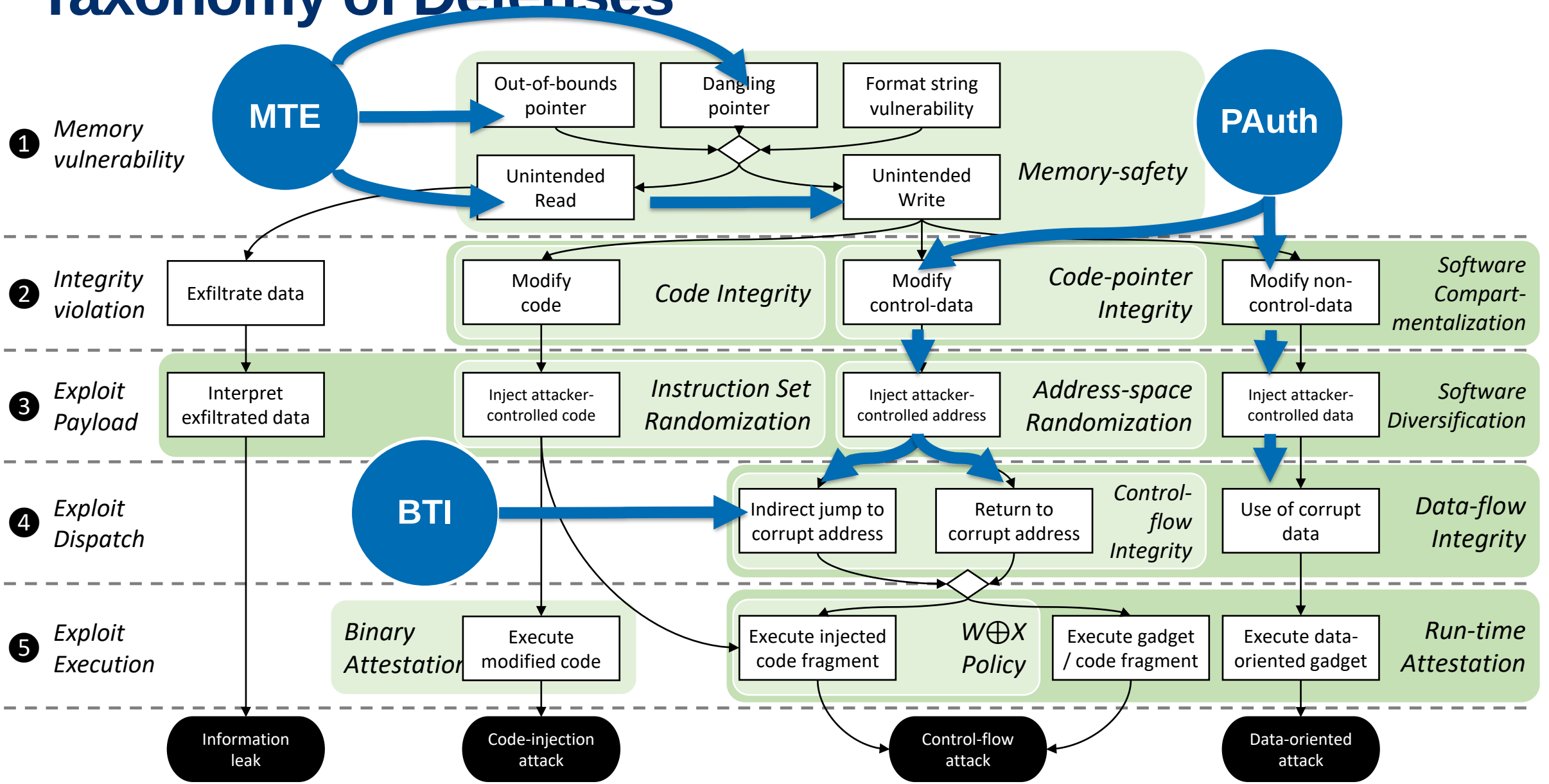
| | |
|---|---|
| **BTI call type branches** | |
| `BLR …` | Indirect function calls |
| `BR <x16\|x17>` | PLT entries and tail calls |
| **BTI jump type branches** | |
| `BR … (except x16\|x17)` | Branches to jump tables |

**BTI Marker Instructions**

| | |
|---|---|
| `BTI <c\|j\|cj>` | Branch Target Identification for c=calls, j=jumps, cj=calls or jumps |
| `BRK` | Breakpoint Instruction |
| `HLT` | Halting breakpoint |
| `PACIASP / PACIBSP` | Create PAC for Instruction address in LR using key A/B and SP as modifier |

ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. Version E.a. July 2019

# Taxonomy of Defenses




Adapted from Szekeres et al., SoK: Eternal War in Memory, IEEE SP (2013)