Aalto University

UNIVERSITY OF WATERLOO

Hardware-assisted Run-time Protection

Thomas Nyman, Hans Liljestrand, Lachlan Gunn, N. Asokan

How to thwart run-time attacks?

Run-time attacks are now routine

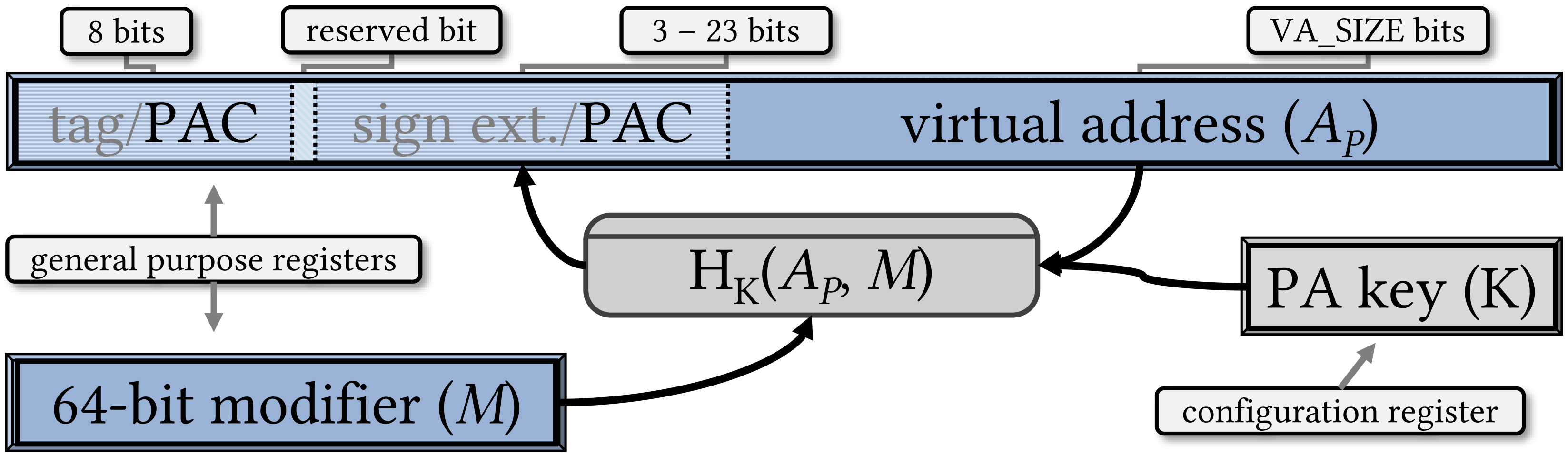
Software defenses incur security vs. cost tradeoffs

Hardware-assisted defenses are attractive

ARMv8.3-A PA – PAC Generation

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

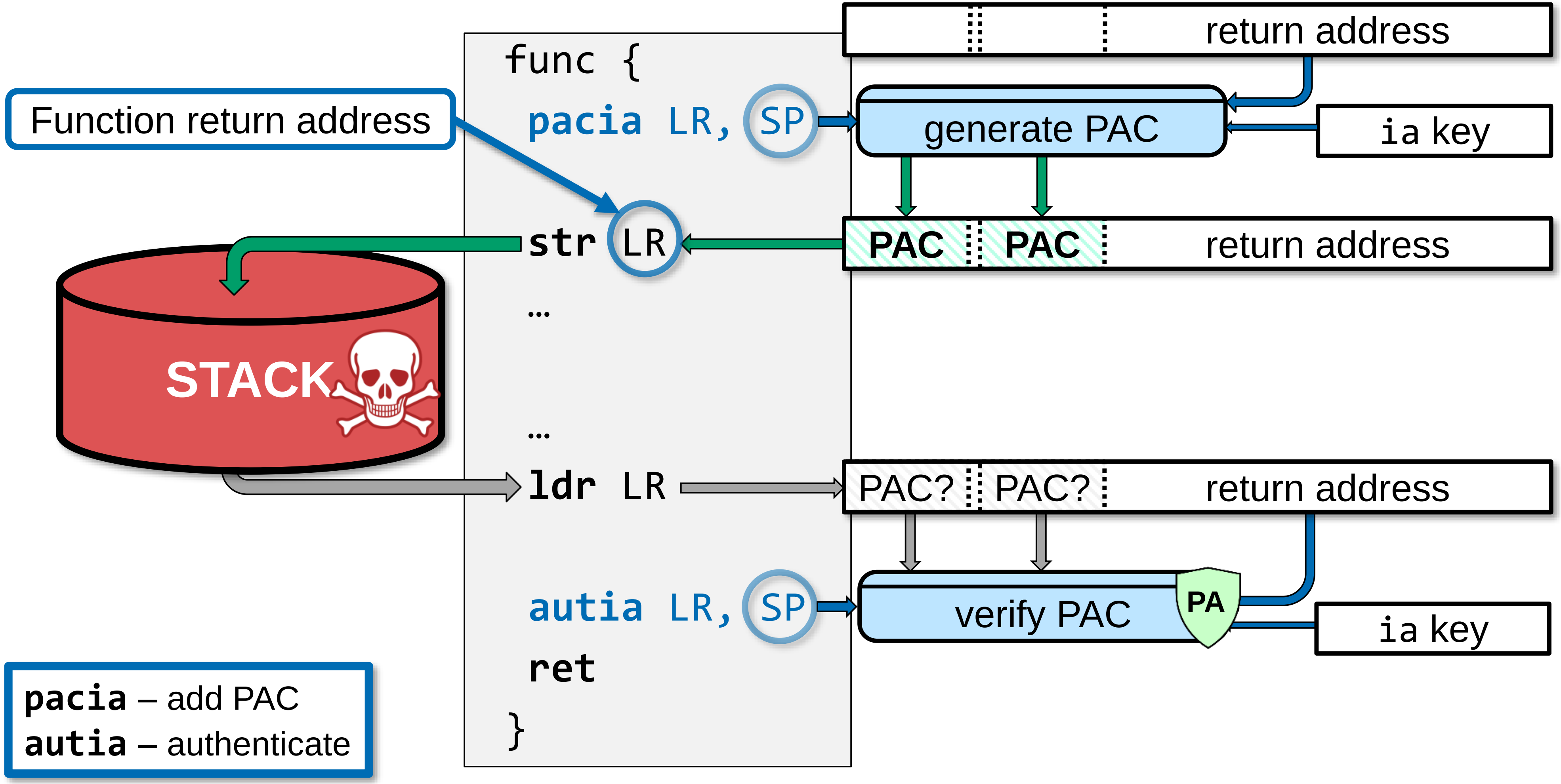
- Keyed, tweakable MAC from pointer address and 64-bit modifier
- PA keys protected by hardware, modifier decided where pointer created and used



ARM. Arm® Architecture Reference Manual Armv8, for Armv8-A architecture profile. version E.a. (2019)

Example: `-msign-return-address`

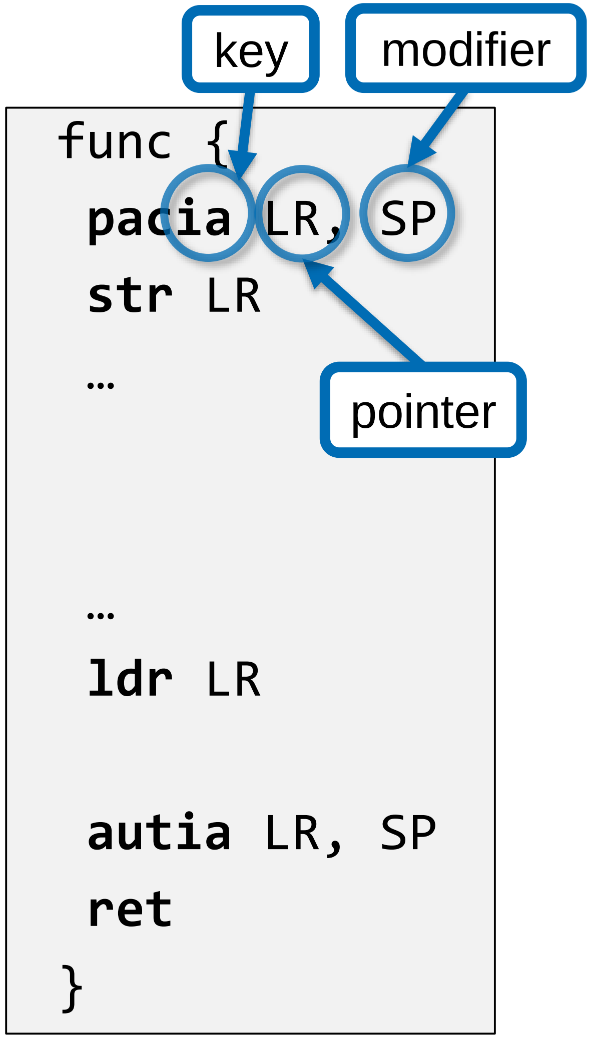
Deployed in GCC 5.0 and LLVM/Clang 7.0



Qualcomm “Pointer Authentication on ARMv8.3” (2017)

PA prevents arbitrary pointer injection

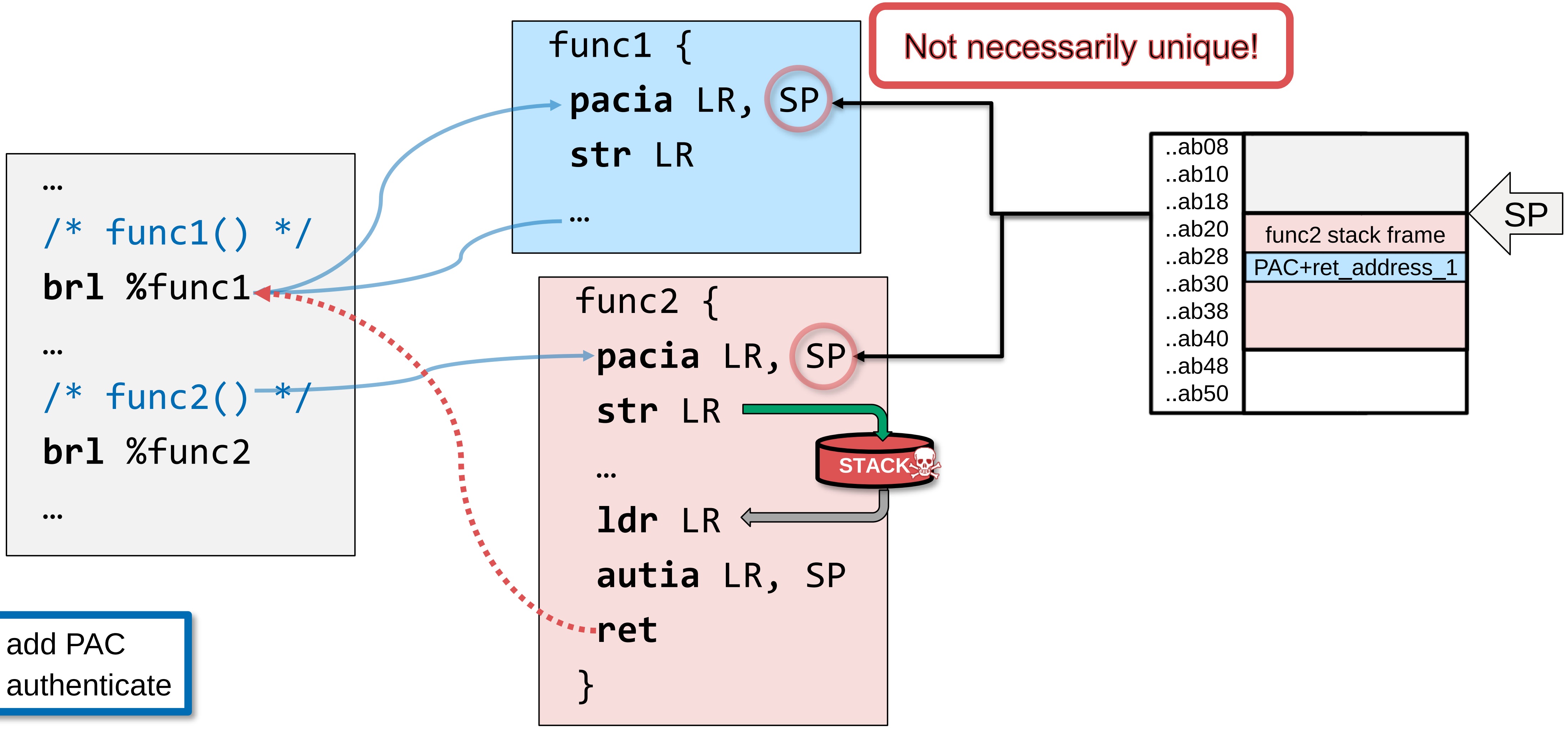
- **Modifiers do not need to be confidential**
  - Visible or inferable from the code section / binary
- **Keys are protected by hardware and set by kernel**
  - Attacker cannot generate PACs



`pacia` – add PAC
`autia` – authenticate

PA only approximates fully-precise pointer integrity

Adversary may reuse PACs



pacia – add PAC
autia – authenticate

[LNWPEA19] PAC it up: Towards Pointer Integrity using ARM Pointer Authentication. USENIX Security (2019)

PA-assisted Run-time Safety (PARTS)

Expands scope of PA protection

- Return address signing
- Code pointer signing
- Data pointer signing

Mitigates pointer reuse by binding

- return addresses to the function definition
- code and data pointers to the pointer type

```
func {
  mov Xmod, SP
  mov Xmod, #f_id, #lsl_16
  pacia LR, Xmod
  …
  mov Xmod, SP
  mov Xmod, #f_id, #lsl_16
  retab Xmod
}
```

pacib – add PAC with instr A-key
retab – authenticate and return

[LNWPEA19] PAC it up: Towards Pointer Integrity using ARM Pointer Authentication. USENIX Security (2019)

Can we do more than PARTS?

PARTS narrows the scope of reuse attacks

- **but cannot completely prevent them**

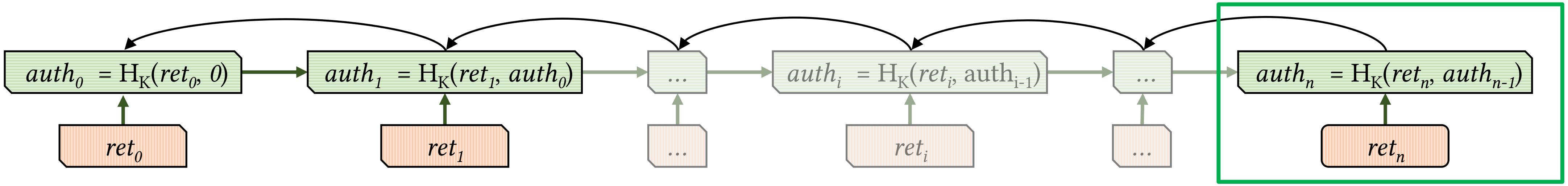
How to optimally minimize scope for reuse attacks?

- **Having unique modifiers often impossible**
- **Static approaches limited to large equivalence classes**

Authenticated Call Stack: high-level idea

Chained MAC of authentication tokens cryptographically bound to return addresses

- Provides modifier (*auth*) bound to all previous return addresses on the call stack
- Statistically unique to control-flow path
  - prevents reuse
  - allows precise verification of returns



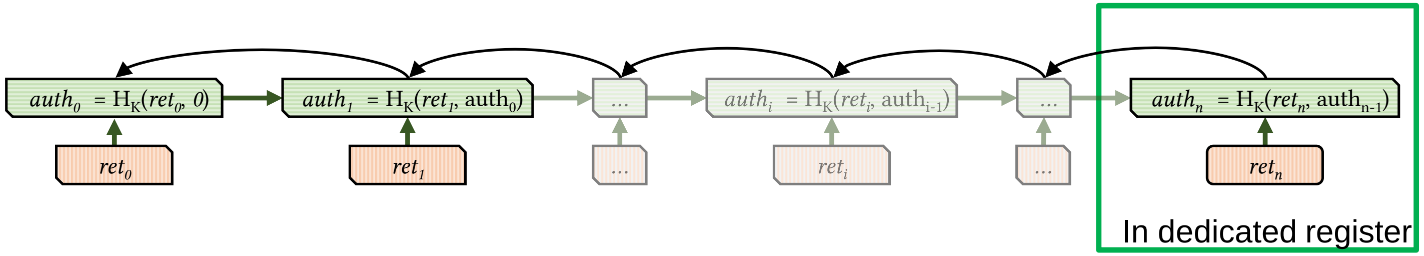
$auth_i$, $i \in [0, n-1]$ bound to corresponding return addresses, ret_i , $i \in [0, n]$, and $auth_n$

[LNGEA19] PACStack: an Authenticated Call Stack preprint (2019)

PACStack instrumentation

- **Generate 16-bit *auth* with `pacib` instruction and embed in PAC-bits**
- **Topmost $auth_n$ is always**
  - Stored securely in dedicated CPU register (`LR`)
  - Passed to callees via the `x28` register

```
prologue:
   str X28, [SP]    ; stack ← aret_{n−1}
   pacib LR, X28    ; LR ← aret_n
function_body:
   ...
epilogue:
   ldr X28, [SP]    ; X28 ← aret_{n-1}' from stack
   autib LR, X28    ; LR ← (ret_n or ret_n*)
   ret
```



Mitigation of hash-collisions: PAC masking

- Challenge**: PAC collisions occur on average after 1.253*$2^{b/2}$ return addresses**
  - For b=16 this is only 321 addresses

- Solution**: Prevent *recognizing* collisions by masking each *auth***
  - pseudo-random mask generated using `pacib(0x0,` $auth_{i-1}$)

Attack	w/o Masking	w/ Masking
Reuse previous auth collision	1	2^{-b}
Guess auth to existing call-site	2^{-b}	2^{-b}
Guess auth to arbitrary address	2^{-2b}	2^{-2b}

Maximum probability of success for different attacks

PARTS & PACStack performance

Functional evaluation

- **On ARM Fast Models 11.4 FVP**

Performance evaluation

- **96board Kirin 620 HiKey board**
- **PA-analog with overhead of 4-cycles**
  - Based on QARMA overhead estimate
  - Uses XOR operations to “sign” pointer

PARTS on nbench-byte-2.2.3

- **Return address protection** **<0.5%**
- **Code pointer integrity** **<0.5%**
- **Data pointer integrity** **~20%**

PACStack on SPEC CPU 2017

- **Without masking** **~0.4%**
- **With masking** **~0.9%**

- Cf. LLVM ShadowCallStack ~0.5%

[A17] The QARMA block cipher family IACR (2017)

How does return-address protection using PA compare with other hardware-assisted approaches?

Intel CET vs. ARMv8.3-A PA

	Intel CET	ARMv8.3-A PA
Return address protection	✓	✓
Indirect branch protection	✓ (coarse-grained)	✓ (PARTS)
Data pointer protection	✗	✓ (PARTS)
Enforcement model	Deterministic	Probabilistic
Immune to pointer reuse	✓	✓ (PACStack)
Memory Overhead	Low to Moderate	N/A
Run-time Overhead	? (likely low)	Low

[LNWPEA19] PAC it up: Towards Pointer Integrity using ARM Pointer Authentication. USENIX Security (2019)
[LNGEA19] PACStack: an Authenticated Call Stack preprint (2019)

Other uses of PA

PA is a general-purpose primitive

PCan - using PA to generate stack-canaries

- **Return address protection already functionally a canary:**
  - Return address corruption due to overflow is detected
  - No reference canary needed
  - Canaries can differ from function to function

- **Reuse still possible, but PCan can be anchored to other schemes**
  - E.g., with PACStack statistically unique canaries for each function call

[LGNEA19] Protecting the stack with PACed canaries SysTEX ’19 (to appear 2019)

Other hardware primitives

Use other emerging hardware primitives for run-time protection?

- **For instance: memory tagging, branch target indication**
- **Can these strengthen each other?**
- **What becomes feasible by combining these primitives?**
- **How do different types of hardware-assistance compare?**
  - ***Is there an optimal set of hardware primitives for new platforms?***

Optimal use of hardware primitives

PA is a powerful security primitive, but others are on the horizon

How to combine them for best trade-off in security, cost, and performance?

https://pacstack.github.io

github.com/pointer-authentication