Aalto University

UNIVERSITY OF WATERLOO

PACStack
an Authenticated Call Stack

Hans Liljestrand[†], Thomas Nyman [‡], Lachlan J. Gunn[‡], Jan-Erik Ekberg[§‡], N. Asokan[†‡]

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

Control-flow attacks

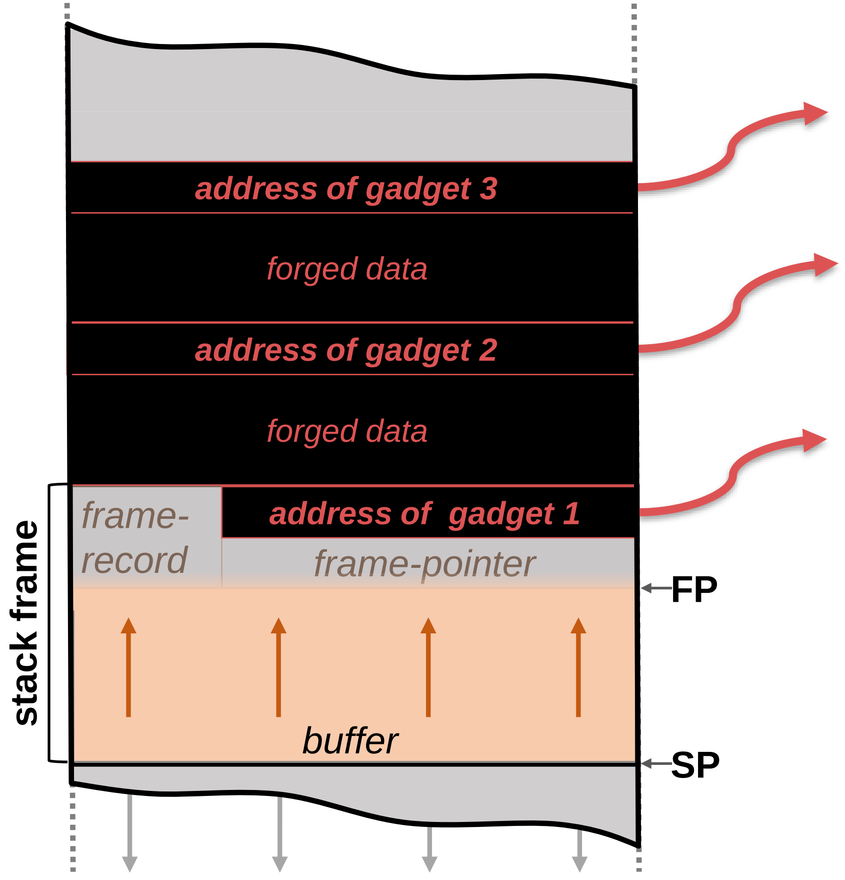
Exploit memory error (e.g. buffer overflow) to:

- Inject shellcode into writable memory (usually stack)
- Corrupts return address to redirect execution flow

Code injection is prevented by modern defenses

Return address corruption still prevalent!

- Code reuse attacks, e.g., ROP



Elias Levy (as *Aleph One*), Smashing the stack for fun and profit, Phrack 7 (1996)
Cowan et al., StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks, USENIX Security (1998)
Szekeres et al., SoK: Eternal War in Memory, IEEE SP (2013)

ARMv8.3-A Pointer Authentication

General purpose hardware primitive approximating pointer integrity

- Ensure pointers in memory remain unchanged
- Uses message authentication codes embedded in pointers

Introduced in ARMv8.3-A specification (2016)

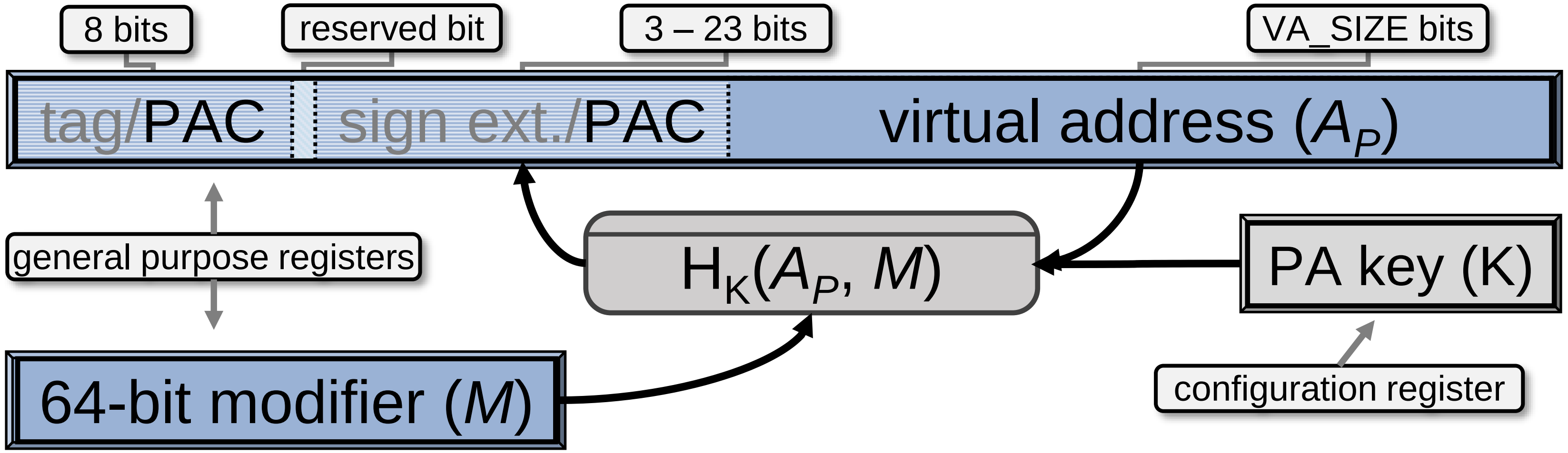
- First compatible processors 2018 (Apple A12 / iOS12)
- Userspace support in Linux 4.21, in-kernel support in 5.7
- Instrumentation support introduced in GCC 7.0 and Clang 8

Apple Inc. "Pointer Authentication – Clang documentation", git (2019)
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

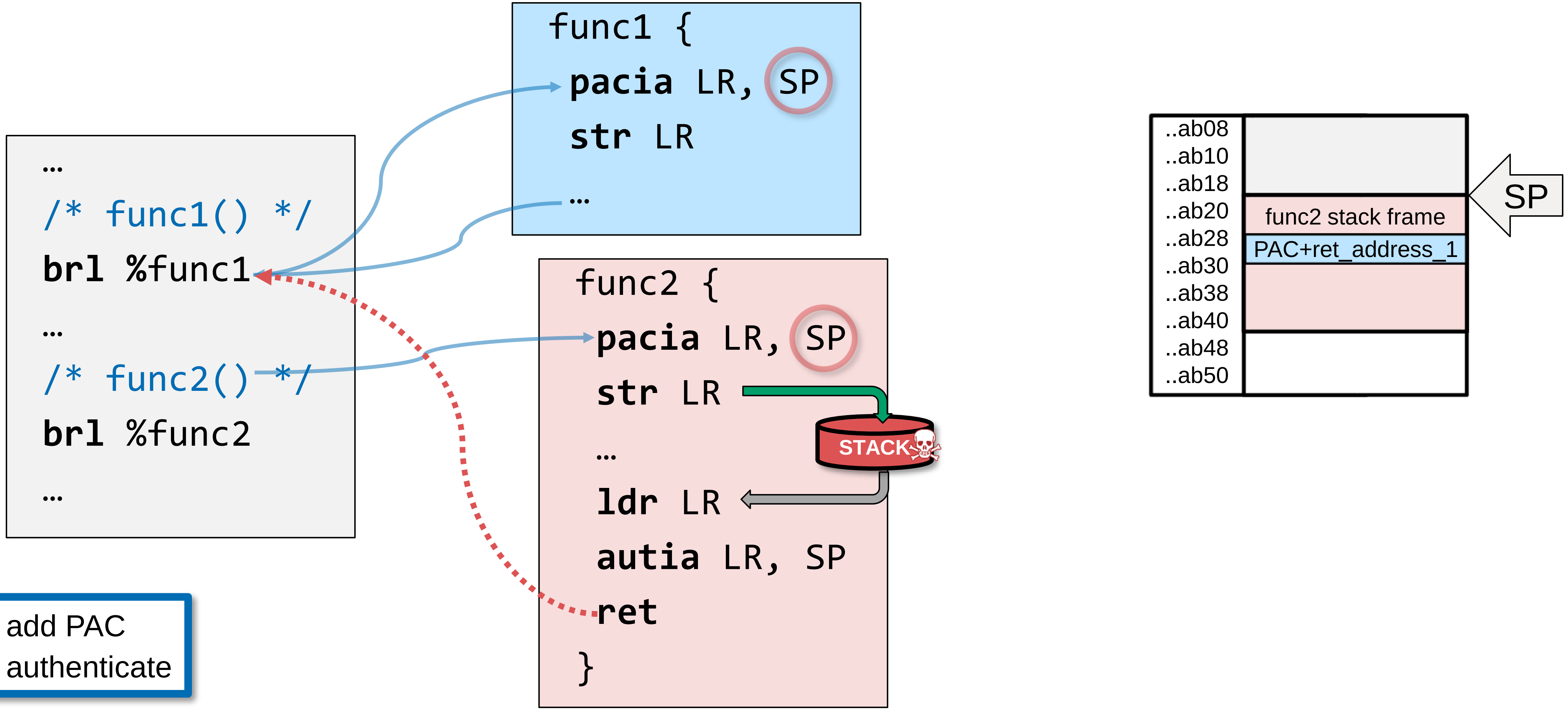
- Keyed, tweakable MAC from pointer address and 64-bit modifier
- PA keys protected by hardware, modifier is decided where pointer created and used



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

Reuse attacks on PA

Adversary may reuse PACs

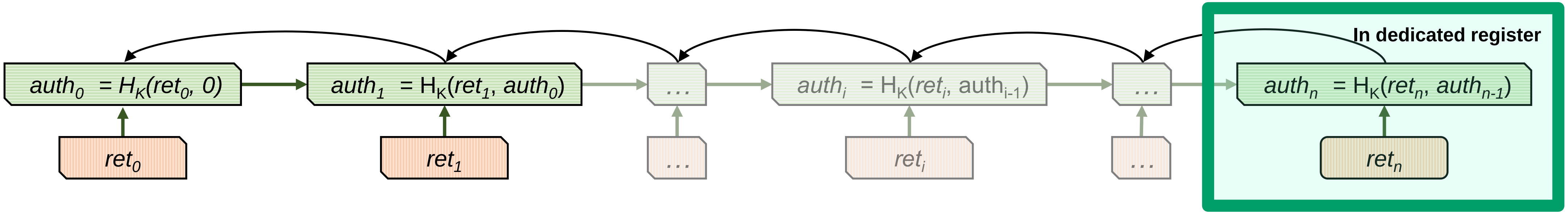


pacia – add PAC
autia – authenticate

PACStack: high-level idea

Chained MAC of cryptographically bound return addresses

- Modifier 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 $auth_n$

Collisions still happen!

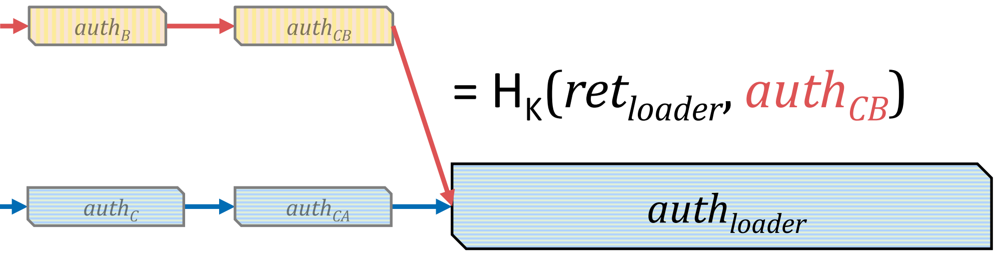
PAC modifiers meant to prevent pointer reuse

But collisions allow reuse

- Registers cannot be directly corrupted
- But values loaded from the stack are vulnerable

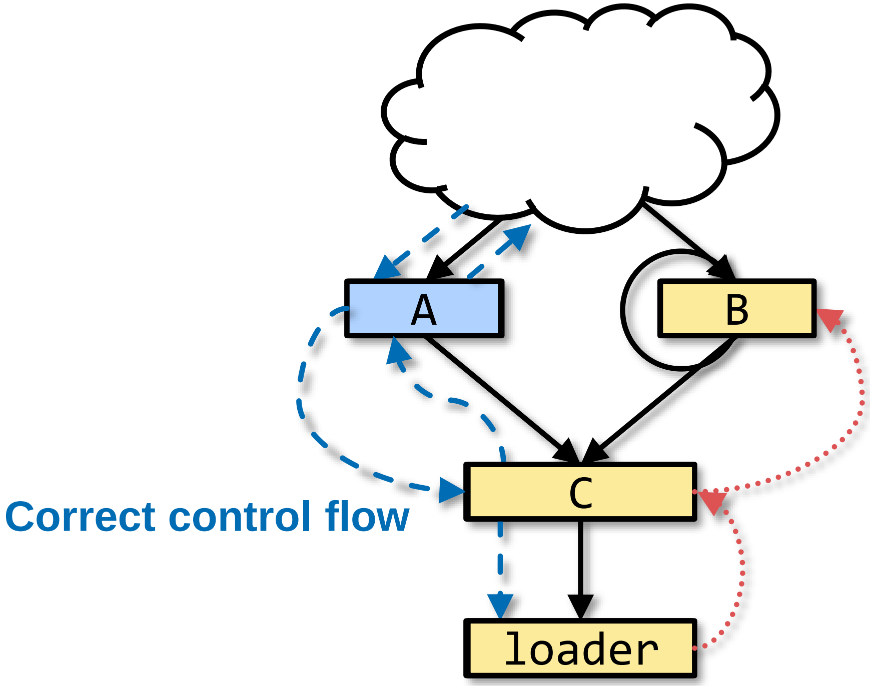
Collisions allow reuse of PAC chain

- Harvest and reuse repeatedly for arbitrary paths



$= H_K(ret_{loader}, auth_{CB})$

$= H_K(ret_{loader}, auth_{CA})$



Mitigation of hash-collisions: PAC masking

Challenge: PAC collisions occur on average after 1.253 * $2^{b/2}$ return addresses

- For $b = 16$, $n = 321$ addresses

Solution: Prevent recognizing collisions by masking each *auth*

- pseudo-random mask generated using `pacia(0x0,` $auth_{i-1}$`)`
- exploitation changes from deterministic to probabilistic

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

Provable security

PA-based schemes can be analyzed as crypto protocols

We prove security reduction for PACStack

- to collision-resistance of PAC primitive

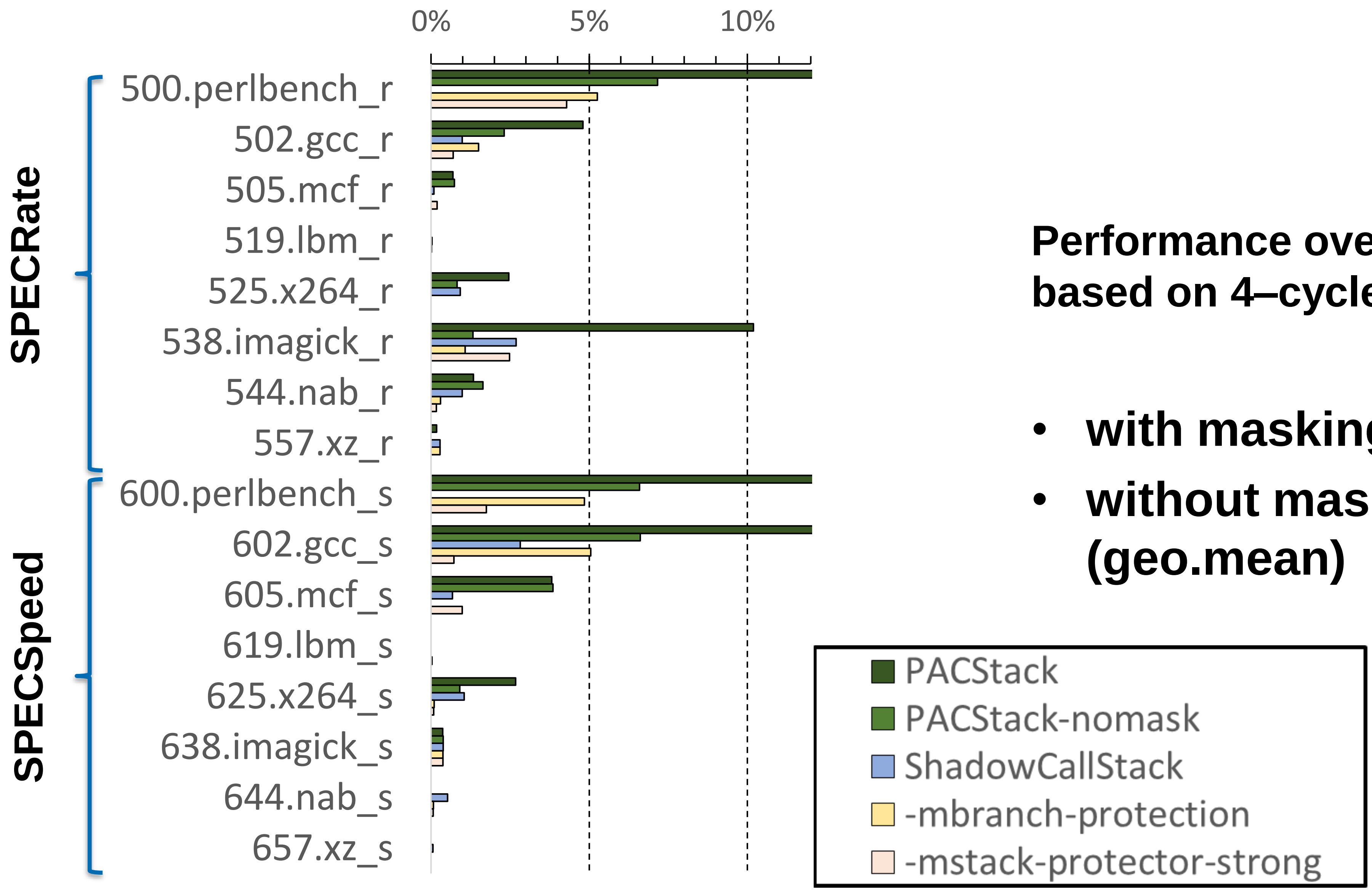
Formal analysis identified vulnerabilities in early design

- Identified collision attack against non-masked PACStack
- Proved masked PACStack secure

$G^{\mathcal{A}}_{\text{ACS}}(1^\lambda, H, C, q)$

$K \xleftarrow{\$} \{0,1\}^\lambda$
/ Give $\mathcal{A}$ q tokens from call-graph traversals.
for $i \in \{1,\ldots,q\}$ **do**
  $p_{1\ldots m+1} \leftarrow \mathcal{A}_{\text{oracle-request}}()$
  / Is the request for a real path through the call-graph?
  if $\exists j : p_j \rightarrow p_{j+1} \notin \text{edges}(C)$ **then**
    return 0
  endif
  $auth_m \leftarrow \mathtt{T}_K(p_m, \mathtt{T}_K(p_{m-1}, \cdots) \parallel p_{m-1}) \parallel p_m$
  $\mathcal{A}_{\text{oracle-response}}(auth_m)$
endfor
$ptr_{\text{jumper}}, ptr_{\text{correct}}, auth_{\text{correct}}, t_{\text{correct}},$
  $ptr_{\text{adv}}, auth_{\text{adv}}, t_{\text{adv}} \leftarrow \mathcal{A}_{\text{ACS-Violation}}()$
/ The substituted masked authenticated return address must be different.
if $ptr_{\text{correct}} = ptr_{\text{adv}} \wedge auth_{\text{correct}} = auth_{\text{adv}}$ **then**
  return 0
endif
/ Does the return pointer authenticate correctly with the adversary's
/ new masked authenticated return address as the modifier?
if $\mathtt{H}_K(ptr_{\text{jumper}}, auth_{\text{correct}} \parallel ptr_{\text{correct}})$
  $\neq \mathtt{H}_K(ptr_{\text{jumper}}, auth_{\text{adv}} \parallel ptr_{\text{adv}})$ **then**
  return 0
endif
/ Did the adversary provide a valid masked authenticated return address?
if $auth_{\text{adv}} = \mathtt{H}_K(ptr_{\text{adv}}, t_{\text{adv}})$
  return 1
endif
return 0

SPEC CPU 2017 benchmarks



Performance overhead measured based on 4–cycle PA-analogue

- **with masking ~ 3% (geo.mean)**
- **without masking ~ 1,2 % (geo.mean)**

Thank you

PACStack provides security comparable to shadow stacks

- **but, probabilistic vs. deterministic**
- **Small overhead without additional hardware requirements**

PA is a general-purpose mechanism

- **Can support novel non-obvious use cases**
- **New emergent properties when combined with other HW?**

hans@liljestrand.dev
@HansLiljestrand

pacstack.github.io

ssg.aalto.fi/research/projects/harp