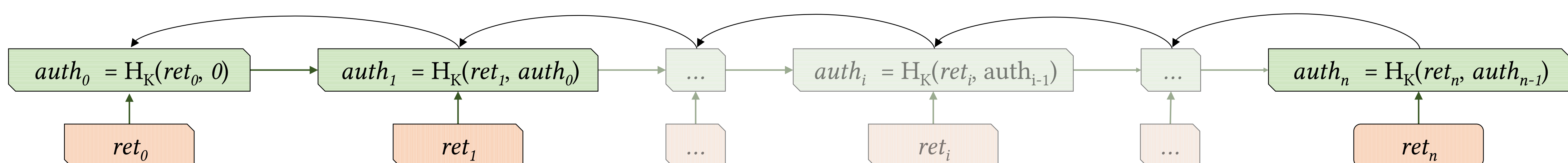
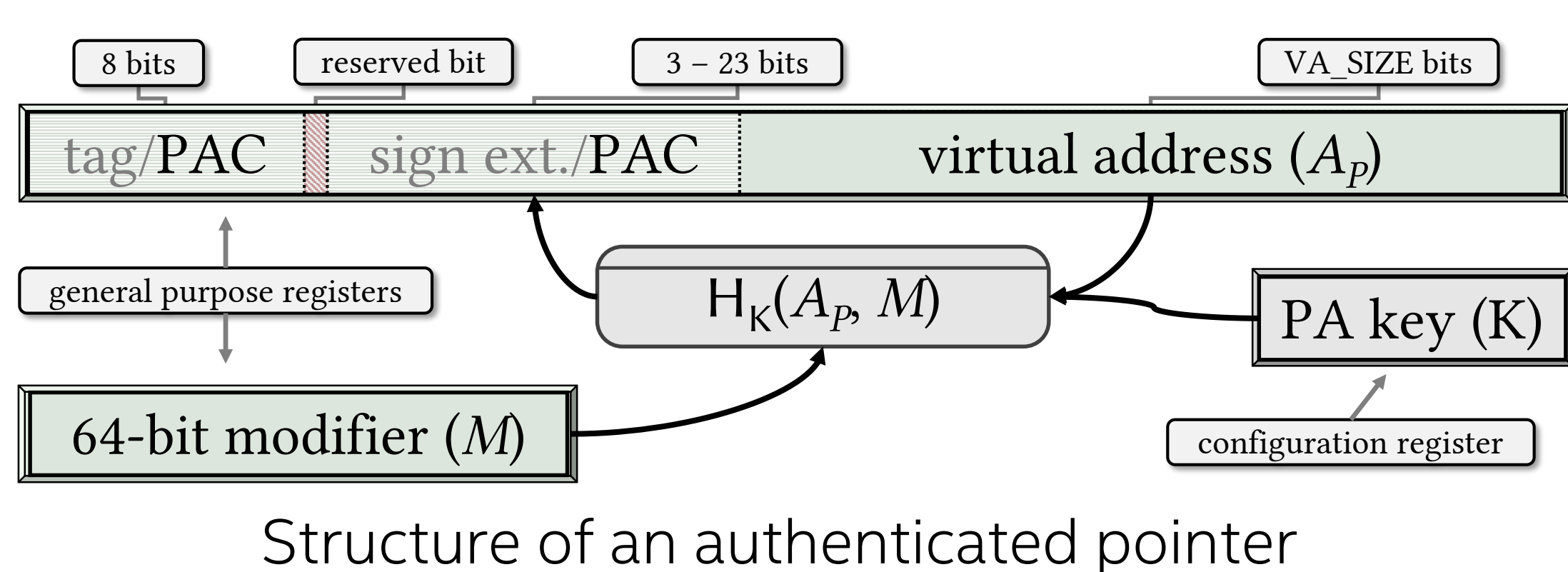


PACStack: Authenticated Call Stack

How can hardware-assistance for *Pointer Authentication* efficiently and precisely verify function return addresses and resist reuse attacks without additional hardware?

•Pointer Authentication deployed in ARMv8.3-A

- General purpose hardware primitive *approximating pointer integrity*
- Adds Pointer Authentication Code (PAC) into unused bits of pointer
- PAC: keyed, tweakable MAC from *pointer address* and 64-bit *modifier*
- PA keys protected by hardware, modifier decided where pointer used
- Vulnerable to **pointer reuse** if modifier is **not unique** to a pointer value



ACS is a chained MAC of tokens $auth_i, i \in [0, n - 1]$ cryptographically bound to corresponding return addresses, $ret_i, i \in [0, n]$, and $auth_n$

Mitigation of hash-collisions: authentication token masking

- **Challenge:** PAC collisions occur on average after $1.253 \cdot 2^{b/2}$ return addresses (e.g., 321 addresses for $b=16$)
- **Solution:** Prevent recognizing collisions by masking each $auth$ with 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

Impact on performance in C-language benchmarks

Estimated performance impact based on PA with QARMA cipher on 1.2GHz ARM core and PA-analogue (4 cycles / PA instruction):

- **0.9%** performance overhead in *SPEC CPU 2017* benchmarks (geometric mean, 0.4% without masking)
- **0.5%** performance overhead in *nbench byte 2.2.3* benchmarks (geometric mean, <0.3% without masking)

High-level idea

- *Authenticated Call Stack (ACS)* is a *chained MAC* of return addresses
- Provide modifier ($auth$) for the return address by *cryptographically binding it to all previous return addresses* in the call stack
- This makes modifier *statistically unique to a particular control-flow path*, preventing reuse and allowing precise verification of returns

ACS implementation using PA: PACStack

Two variants:

1. Generate 32-bit $auth$ with `pacga` instruction and store on stack
 2. Generate 16-bit $auth$ with `pacib` instruction and embed in PAC-bits
- Topmost $auth_n$ always stored securely in dedicated CPU register

Comparison: ACS / PACStack vs. shadow stacks

Shadow stacks are precise, but have drawbacks:

- Software shadow stacks suffer from **large performance overheads**
- A *parallel shadow stack / dedicating a register* increase performance, but leave shadow stack **vulnerable if its location in memory is known**
- Hardware shadow stacks are **efficient** and **secure**, but **require dedicated, single purpose support** and **isolated / integrity protected memory**

ACS / PACStack provide *probabilistic guarantees*, but has benefits:

- Can be **instantiated with any MAC**, e.g., *hardware-assisted utilizing PA*
- Very **efficient** when utilizing **hardware-assisted primitives**
- **No isolated / integrity protected memory** (beyond single register)

Generalizing ACS to other use cases

Provides efficient authenticated stack using ARM PA that can be used for:

- protecting other stack data, e.g., frame pointer or read-only variables
- frame-by-frame unwinding of the call stack in C++ exceptions
- reusable library for protecting other critical data structures (e.g., in kernel code, language runtime, applications etc.)

