arm

Experimenting with the AArch64 Pointer Authentication (Pauth) ABI on bare-metal

Peter Smith
31/01/2026

Arm v8.3-A Pointer Authentication in a nutshell

signed_pointer = **sign**(raw_pointer, key, discriminator);

PAC<key> <Xd>, <Xn>

raw_pointer = **auth**(signed_pointer, key, discriminator);

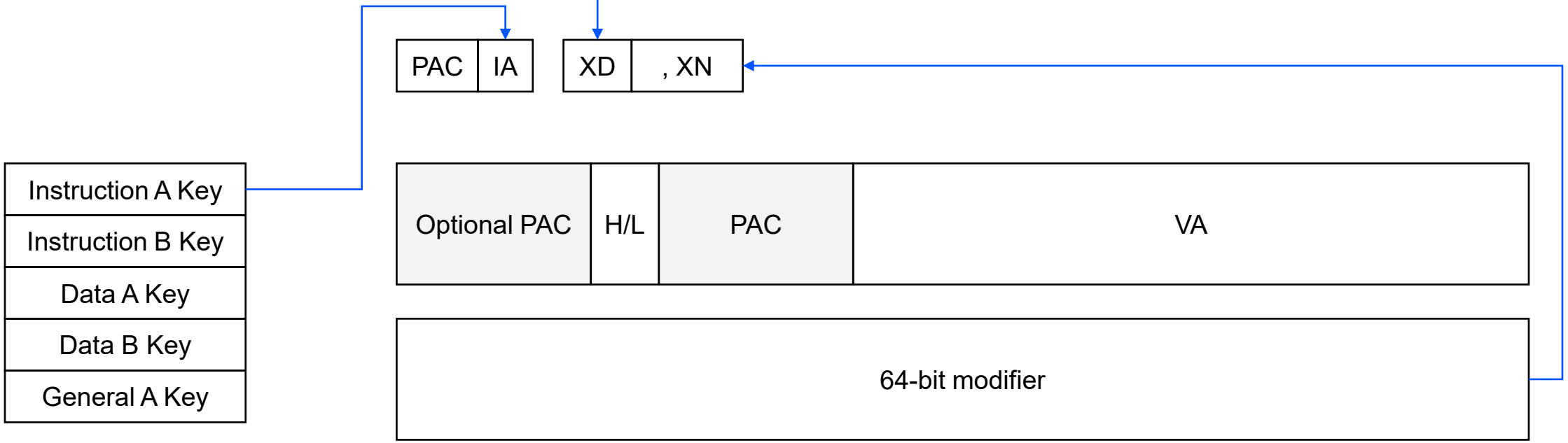
AUT<key> <Xd>, <Xn>

- The key is a shared secret; the discriminator is local to the pointer.
- Which pointers are signed, and how they are signed (key, discriminator) is called a signing schema.
- -mbranch-protection=standard (including pac-ret) is an example signing schema that protects the return address.

Armv8.3-A Pointer Authentication Implementation

Pointer in AArch64

Signed Pointer in AArch64



Why am I talking about PAuthABI?

- Pointer Authentication on ELF platforms is currently restricted to protecting the return address
  - Can be deployed on all AArch64 machines and operating systems.
- PAuthABI has existed as a specification for about 5 years
  - ELF implementation of Apple's Arm64E ABI.
  - PAuthABI protects all code-pointers but requires Arm v8.3-A hardware and an ABI break.
- Upstream LLVM has now got an implementation for testing but there's no public target deploying it.
- Bare-metal has fewer ABI and hardware constraints than a Linux Distribution.
- Would like to see PAuthABI have a production ready target.

PAuthABI extends pointer authentication to code-pointers

```
struct B final {
  virtual int f();
};

int use_vtable(B* b) {
  // load vtable for B from b
  // load address of B::f from
  // vtable for B
  // indirectly call B::f
  return b->f() + 10;
}

// clang++ -target=aarch64-Linux-
pauthtest -mcpu=cortex-a510
```

```
use_vtable:
  PACIBSP
  STP       x29,x30,[sp,#-0x10]!
  MOV       x29,sp
  LDR       x8,[x0,#0] // x8 = vtable ptr
  MOV       x9,x0 // x9 = vtable ptr address
  MOVK      x9,#0x7125,LSL #48 // discriminator
  AUTDA     x8,x9
  LDR       x9,[x8,#0]
  MOV       x17,x8 // Address of vtable entry
  MOVK      x17,#0x2a8e,LSL #48 // discriminator
  BLRAA     x9,x17 // Combined branch and aut
  ADD       w0,w0,#0xa
  LDP       x29,x30,[sp],#0x10
  RETAB     // Combined return and aut
```

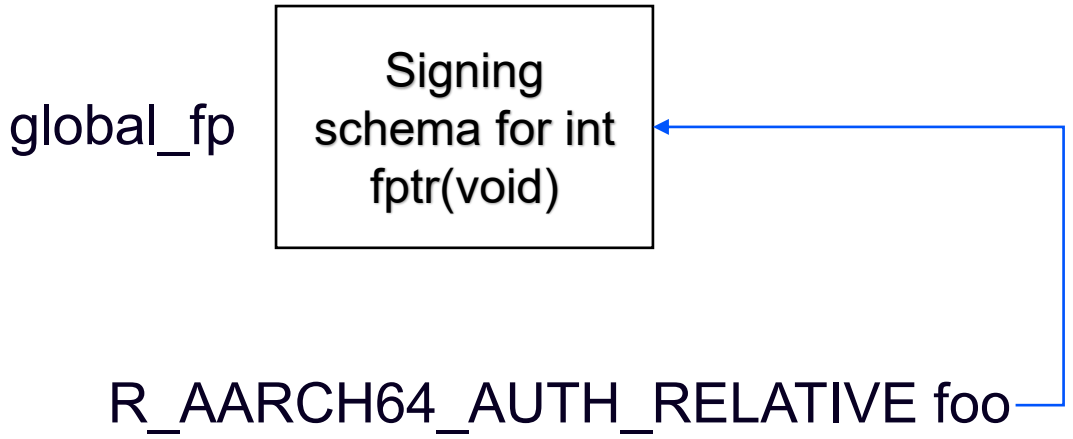
PAuthABI in upstream LLVM

- Experimental test target environment PAuthTest with Linux OS
  - `--target=aarch64-linux-pauthtest`
  - Synonymous with `--target=aarch64-linux-gnu -mabi=pauthtest`
- Target is experimental as any change to the signing schema is an ABI break
  - No public ELF platform has committed to an ABI yet.
- Header file `ptrauth.h` provided for manual control of signing schema
  - Recommended for C-Style pointer to functions as default signing schema accounts for anything permitted by the C-standard.
- Using it requires compatible runtime libraries and dynamic linker support.

Dynamic linker support for static initialization of code pointers

```
extern int foo(void);
typedef int fptr(void);
// global_fp initialized to foo
// prior to entering main
fptr* global_fp = &foo;

int main() {
  // expects global_fp to be signed
  return global_fp();
}
```



- AUTH variant dynamic relocations sign result after traditional relocation operation.
- Signing schema for pointer in the place being relocated.

Trying out PAuthABI in Linux Userspace

- Access Softek have provided a statically linked musl based Linux toolchain.
- Uses a fork of llvm-project and a fork of musl with dynamic linker support.
- Executables produced can be run on an armv8.3-a system or via QEMU user-mode emulation.
- Link in references slide to toolchain build-scripts
  - Script uses an Ubuntu 24.04 LTS container, may need to alter if your distro is older or newer.
  - Produces a squashfs that can be mounted, examples use `/opt/llvm-pauth`

```
/opt/llvm-pauth/bin/clang++ -target aarch64-linux-pauthtest -march=armv8.3-a \
  -Wl,--dynamic-linker=/opt/llvm-pauth/aarch64-linuxpauthtest/usr/lib/libc.so \
  -Wl,--rpath=/opt/llvm-pauth/aarch64-linux-pauthtest/usr/lib \
  hello-world.cpp -o hello-world

qemu-aarch64 hello-world
Hello World!
```

Experimenting with PAuthABI on bare-metal

- Hardware and ABI requirements are challenges for PAuthABI adoption on existing platforms.
- Bare-metal systems, or firmware for a larger system may be an easier target for deployment.
- Can we build an embedded toolchain that supports PAuthABI on `qemu-system-aarch64` ?
  - Make life easier with `-fno-exceptions -fno-rtti` and no position independence.
- Shopping list:
  - PAuthtest support in the bare-metal driver `aarch64-none-pauthtest`.
  - Compiler-rt, llvm-libc and libc++ runtime with `aarch64-none-pauthtest` support.
  - Support code to initialize pointer authentication keys.
  - Linker script to add linker defined symbols to `.rela.dyn` bounds
  - A relocation resolver to resolve `R_AARCH64_AUTH_RELATIVE` relocations.
  - Multilib selection of runtime from `aarch64-none-pauthtest`.
- Link to fork of llvm-project with build-scripts in references.

Experience

- Baremetal driver aarch64-none-elf does not support PAuthTest or -mabi=pauthtest
  - Add and adapt the existing Linux driver code.
- LLVM libc has some inline assembly that needed altering for PAuthABI
  - `asm volatile("bl %0" : : “X"(LIBC_NAMESPACE::do_start));`
    - In PAuthABI do_start must be accessed indirectly, replace `bl` with `blr`, “X” with “r”.
  - `asm("B %0" : : “X"(GenericException_Handler));`
    - Replace `B` with `BR`, “X” with “r”.
- Multilib can be set up to use the `aarch64-none-pauthtest` target.
- Can use the AUTH variant relocations, using linker defined symbols to find the relocations.

```
.rela.dyn : {
                PROVIDE(__rela_dyn_start = .);
                *(.rela.dyn .rela.dyn.*)
                PROVIDE(__rela_dyn_end = .);
            } >flash AT>flash :text
```

Processing dynamic relocations at startup

- `R_AARCH64_AUTH_RELATIVE` with operation
  - `SIGN(DELTA(S) + A, SCHEMA(*P))`
- Signing schema stored in the place of the relocation *P
  - `Modifier` calculated from `address diversity`, `discriminator` and place `P`.
- Store result of PAC<key> xd, xn into contents of place *P
  - `xd = DELTA(S) + A`
  - `xn = modifier`

63	62	61:60	59:48	47:32	31:0
Address diversity	reserved	Key 00 IA 01 IB 10 DA 11 DB	reserved	discriminator	Reserved for addend (when doing RELR compression)

Next Steps for PAuthABI

- PAuthABI support in a chicken-and-egg situation
  - Projects can’t use it if there’s no toolchain.
  - Toolchain providers won’t do the work to support PAuthABI unless real projects want to use it.
- Likely that the signing schemas coalesce into a small number of supportable variants
  - With an official ELF target bare-metal, upstream support for bare-metal PAuthABI is feasible.

References

- Clang documentation on Pointer Authentication
  - https://clang.llvm.org/docs/PointerAuthentication.html
  - Required reading for anyone building a production system!
- Access Softek's Linux musl PAuth Toolchain
  - https://github.com/access-softek/pauth-toolchain-build-scripts
- LLVM fork for a proof-of-concept embedded toolchain
  - https://github.com/smithp35/llvm-project/tree/pauthabi
- LLVM Dev Meeting 2024 presentation
  - Adding Pointer Authentication ABI support for your ELF platform
  - https://www.youtube.com/watch?v=bytWm7BzJVE
- PAuth ABI Extension to ELF for the Arm 64-bit Architecture (AArch64)
  - https://github.com/ARM-software/abi-aa/blob/main/pauthabielf64/pauthabielf64.rst
- Arm Toolchain LLVM libc support code
  - https://github.com/arm/arm-toolchain/tree/arm-software/arm-software/embedded/llvmlibc-support

Backup: Limitations of proof of concept

- Runs on QEMU virt machine
  - `qemu-system-aarch64 -machine virt -cpu max -semihosting -kernel hello.elf  -nographic -machine virtualization=on`
- Tested on some simple C and C++ programs that dereference global code-pointers and vtables.
- PAC keys hardwired to test values
  - Would need to have a source of randomness to have different keys each boot.
- No support for relocation read-only for .data.rel.ro
  - If GOT used it would need to be signed.
- LLVM libc setjmp and longjmp may need additional hardening.
- Exceptions and RTTI support

Backup: Pointer Authentication using PAC RET

```
// return address in x30 on function entry
f:
  PACIASP  // hint space encoding of PACIA x30, sp
  STP      x29,x30,[sp,#-0x20]!
  …
  LDP      x29,x30,[sp],#0x20
  AUTIASP  // hint space encoding of AUTIA x30, sp
  RET
```

- Designed for minimal ABI impact.
- Enabled with -mbranch-protection=[standard|pac-ret]
- Hint space encoding means code can run on all AArch64 machines.
- Enabled by default in some Linux distributions.

arm

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm