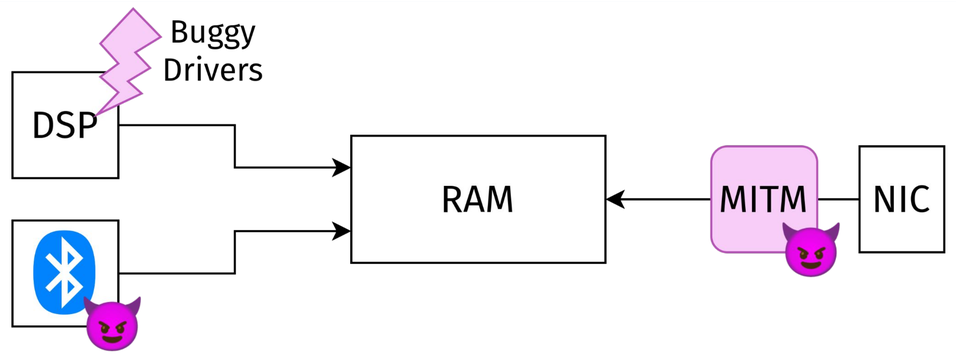
De/centralized capabilities for DMA protection

Samuel W. Stark, A. Theodore Markettos, Simon W. Moore
Department of Computer Science and Technology, University of Cambridge
sws35@cam.ac.uk

UNIVERSITY OF CAMBRIDGE
Computer Science & Technology

DMA protection is necessary

Modern systems have **many peripherals** made by **different vendors** with **opaque firmware**. Peripherals may be **buggy or malicious**, and they cannot be left unchecked**.** As systems get bigger and more peripherals are added, the problem is further complicated. We need **dynamic, fine-grained** protection that we can **create and revoke quickly**.



IOMMUs have significant limitations

Clunky Access Control Lists

The IOMMU is an industry-standard form of DMA protection, allowing **untrusted peripherals** to access **host RAM** while sandboxed in a **virtual address space**. Every access a peripheral attempts is checked by walking a **page table**.

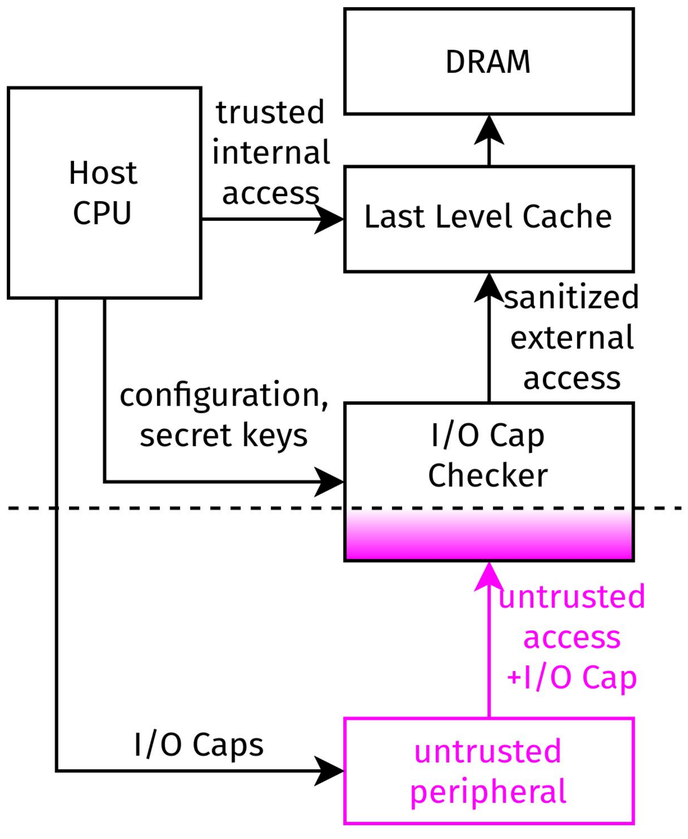
- The IOMMU offers **coarse-grained memory protection (4KiB or larger)**
- IOMMU entries are **non-transferable**
  - Peer-to-peer sharing requires collaboration with the host
- IOMMU lookups are **slow**
  - So need to be cached in an IOTLB
- DMAs have **low locality**
  - So caching can be ineffective
- IOTLB invalidations are **slow**
  - So we **defer** them and batch many together
- Deferred invalidations are **unpredictable**
  - Pages can be exposed for **10ms after requesting invalidation** on Linux, with **no way to avoid buffer reuse** while exposed

IOCaps are the solution!

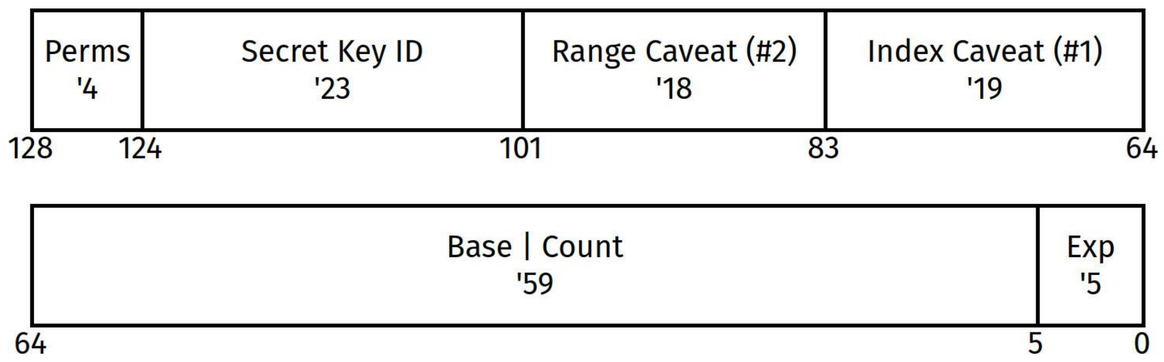
Capabilities strike again

An IOCap is a form of **Macaroon**, or **capability** - an unforgeable token of authority - verified with **cryptography**. Peripherals are only given IOCaps for the regions of memory they absolutely need, and **cannot access host RAM without a matching capability**. Peripherals can attach **caveats** to shrink permissions further, and pass those reduced caveats on to other peripherals **without host intervention**.

- IOCaps offer **fine-grained memory protection (1-byte)**
- IOCaps are **transferable** and **shrinkable**
  - Caveats are verified with a **signature chain**
- IOCap checking is **fast**
  - Binary format optimized for decoding
  - AES-128 signature verification
  - Caching is unnecessary, locality is irrelevant
- IOCap invalidations are **instantaneous**
  - IOCaps are verified with a **secret key** stored in the checker
  - IOCaps can be revoked in **groups** by forgetting their secret key

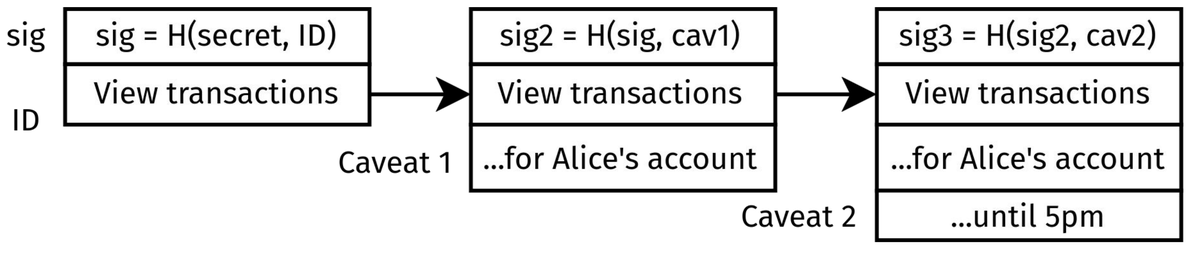


The IOCap usage model. Driver software asks the OS to allocate secret keys and generate IOCaps. Peripherals are given IOCaps to grant access to different ranges of RAM, and must present them with each attempted DMA access. The combined signature & plaintext prevent tampering.



The 128-bit IOCap text format. The first 64-bits encode an initial resource, with up to 1KiB granularity, which can be refined to 1B granularity using the caveats. This must be accompanied with a 128-bit signature (not shown), which is the end result of an hash chain starting at the **secret key** ID-d by the capability, similar to Macaroons (shown below).

IOCaps are verified by recomputing the chain of hashes, starting at the secret key which only the IOCap Checker unit knows. An IOCap with a valid signature must have been generated by this unit in the past, and thus must be valid.



Existing Support

- High-throughput IOCap Checker
  - Secret key manager
  - Format decoder
  - AES signature checker
- IOCap-over-AXI hardware protocol
- `libccap` C library for IOCap generation
- Drop-in `virtio` protocol integration
- Emulated QEMU devices
- CheriBSD/FreeBSD drivers
- Cheri/FreeRTOS integration

Future Work

- Implement protection against reusing exposed buffers
- Test revocation and grouping strategies
- Profile latency and throughput impact

```
virtionet_init(52:54:00:12:34:56)


IP Address: 10.0.2.15
Subnet Mask: 255.255.255.0
Gateway Address: 10.0.2.2
DNS Server Address: 8.8.8.8


cap: [v: 1 | f: 0 | sealed: 0 | addr:
0x0000000010007000 | base: 0x0000000010007000 |
length: 0x0000000000001000 | offset:
0x0000000000000000 | perms: 0x000000000000000d |
otype: 0xffffffffffffffff]
virtio_queue_init_vq negotiated size 1024 for
queue 0
virtio-iocap: global keys already set up
virtio-iocap: Wrote queue_iocap text and
signature to device
virtio-iocap: stats gw  20 bw   0 gr  16 br   0
sector: 0x0
virtio-iocap: virtio_fill_desc from virtio addr:
00000000884806f0 len: 00000010 flags: 1 next: 1
success
virtio-iocap: virtio_fill_desc from virtio addr:
00000000824b61f0 len: 00000200 flags: 3 next: 2
success
virtio-iocap: virtio_fill_desc from virtio addr:
00000000822b06cf len: 00000001 flags: 2 next: 0
success
virtio_queue_notify - device 0x824a84c0, vq
```

CheriFreeRTOS using IOCaps over `virtio`

Is this just CHERI?

Two capability systems, alike in dignity?

IOCaps may seem similar to CHERI, an architecture model which uses capabilities inside the CPU to protect access to virtual memory. CHERI uses **tag bits** to indicate capabilities in registers and memory, and CHERI capabilities **can only be manipulated and stored by trusted instructions** in the CPU - enforcing the unforgeability and monotonicity properties that make a capability a capability.

IOCaps are simply data, and **do not require tag bits** for identification. They are designed to **allow untrusted actors to manipulate and store** them while maintaining those properties, which is verified by the signature and format. The trusted base only stores the secret keys. By combining decentralized storage and manipulation with centralized secret keys, IOCaps are de/centralized.

This work was supported by the University of Cambridge Harding Distinguished Postgraduate Scholars Programme.