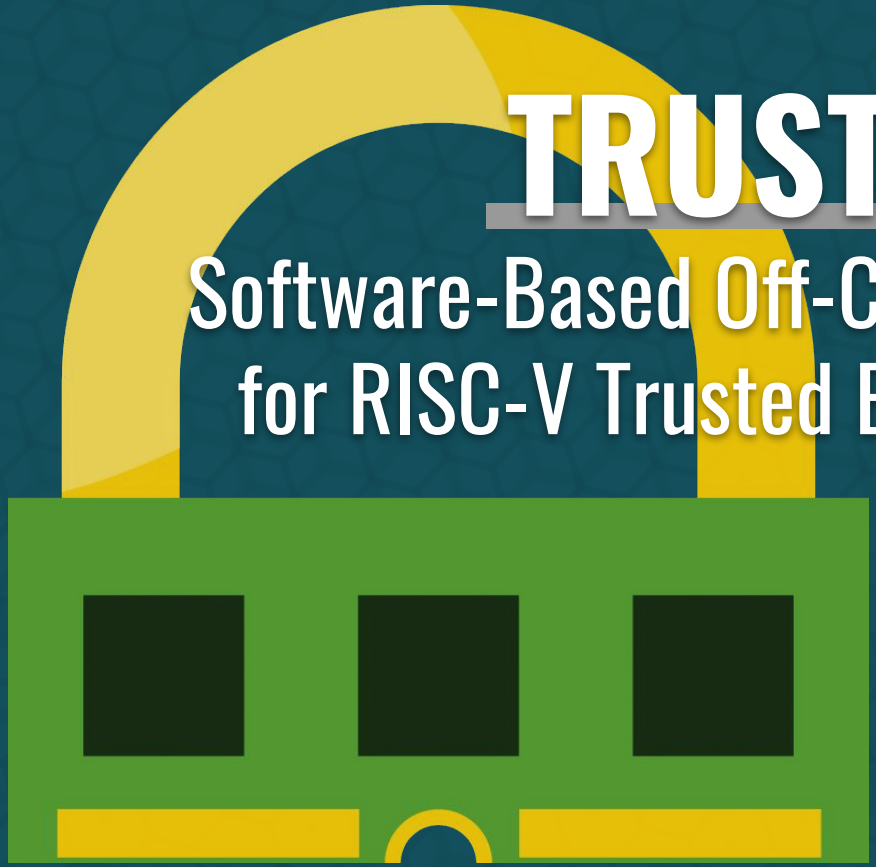


TRUSTED MEMORY

Software-Based Off-Chip Memory Protection for RISC-V Trusted Execution Environments

Gui Andrade
Dayeol Lee
David Kohlbrenner
Krste Asanović
Dawn Song

University of California, Berkeley

TRUSTED MEMORY?

Securing data/code against snooping/reverse-engineering

Cryptographic keys

Proprietary algorithms

Biometric data

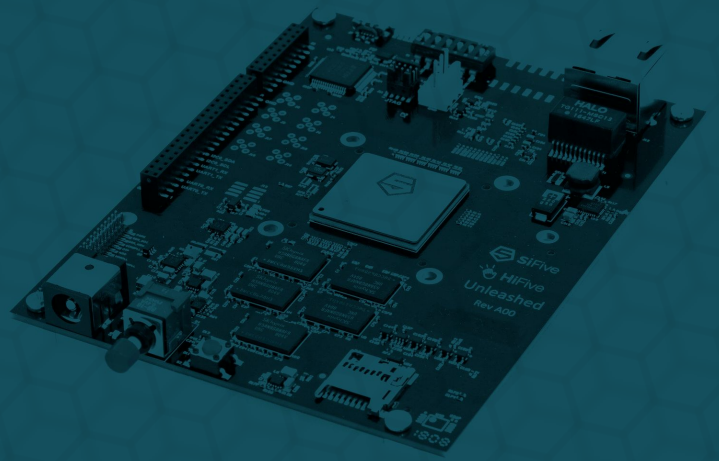
Personally identifiable information

Code obscurity (defense-in-depth)

TRUSTED MEMORY?

Two security domains of system memory: *on-chip* and *off-chip*.

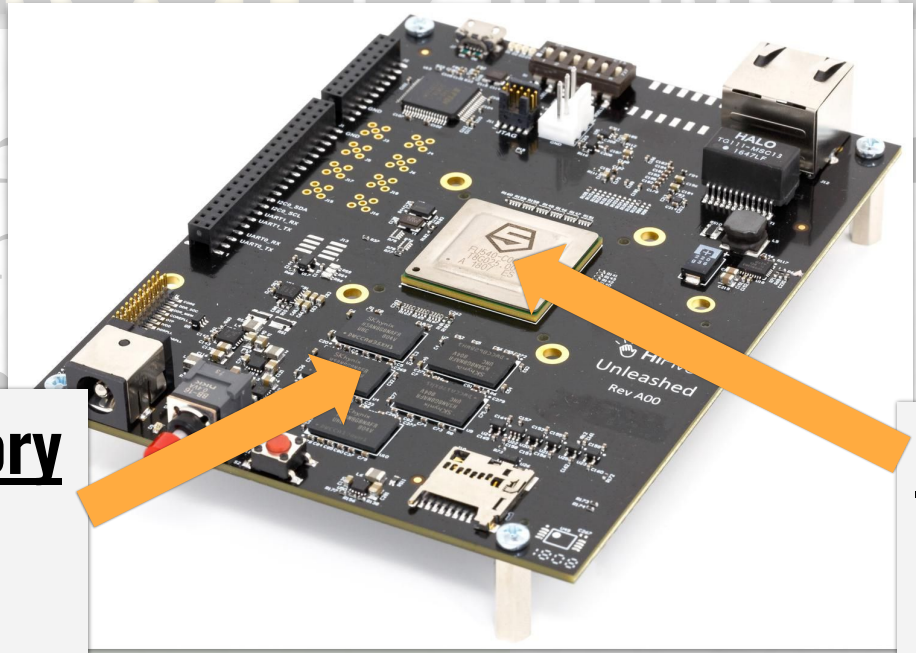
On-chip **presumed secure**,
off-chip not quite



TRUSTED MEMORY?

Two secur… memory:
on-chip ar…

On-chip m…



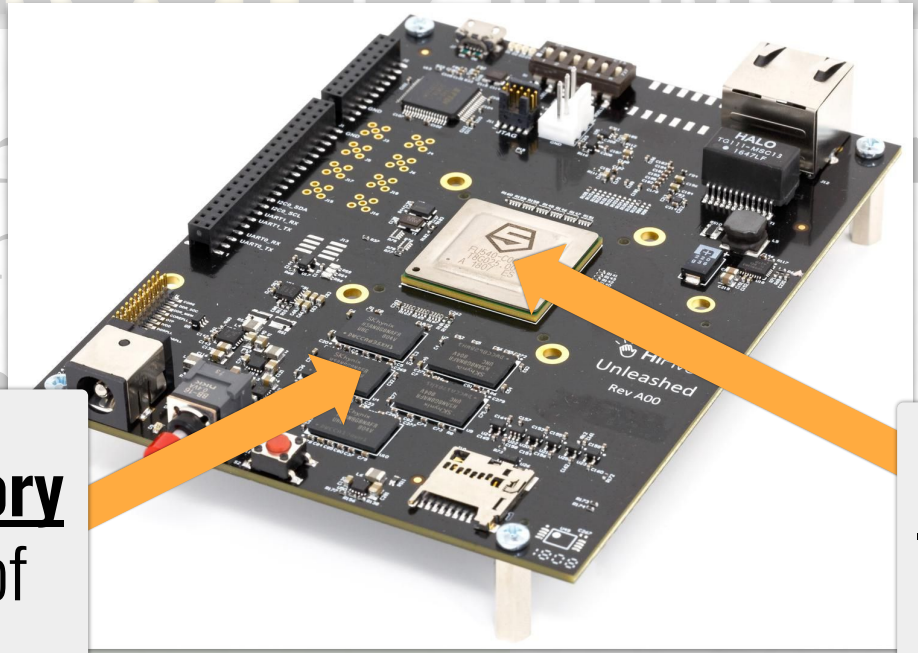
Off-chip memory
a determined hobbyist (!) could compromise

On-chip memory
extremely difficult (expensive) to compromise

TRUSTED MEMORY?

Two secur… memory:
on-chip ar…

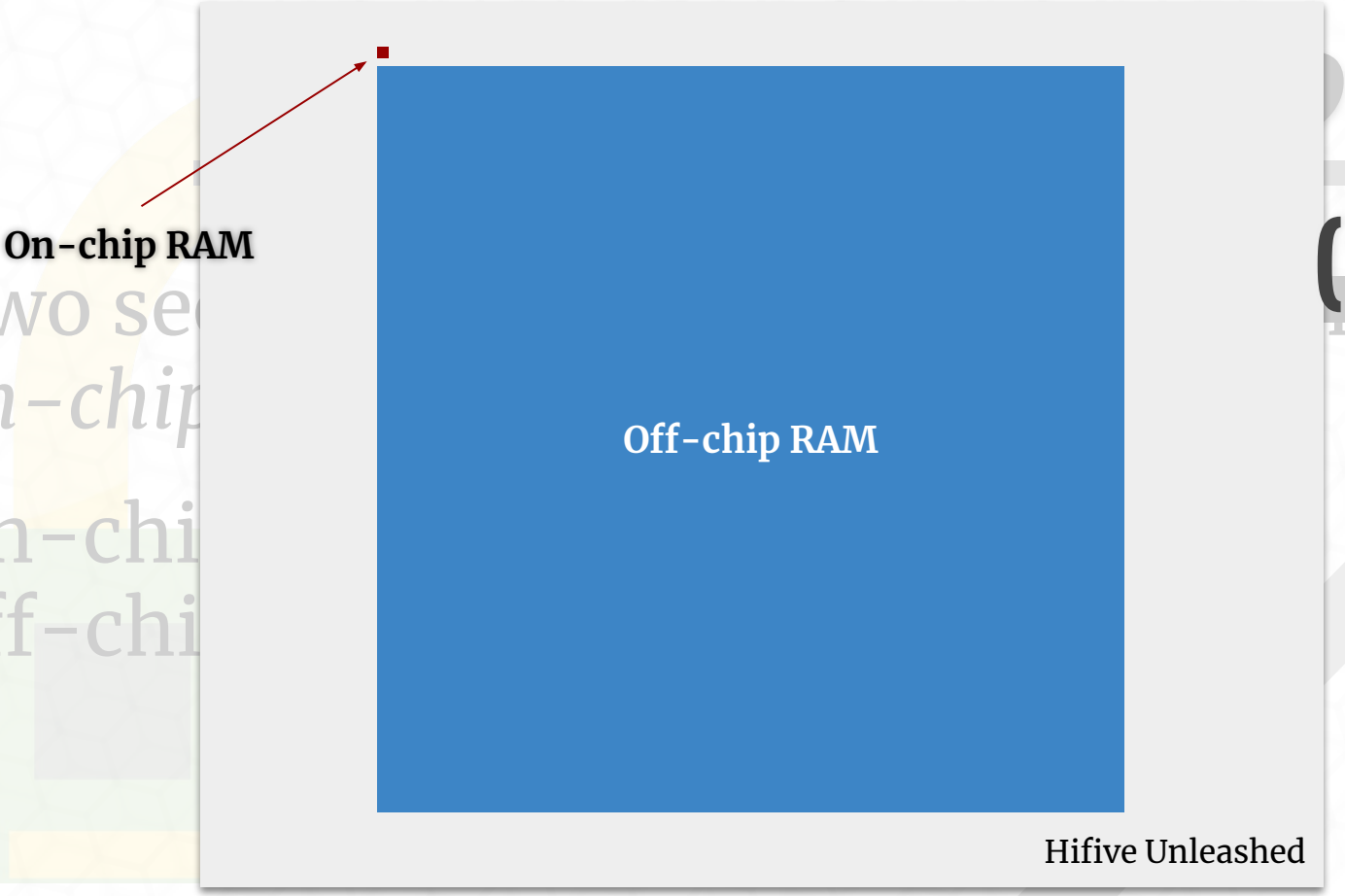
On-chip m…



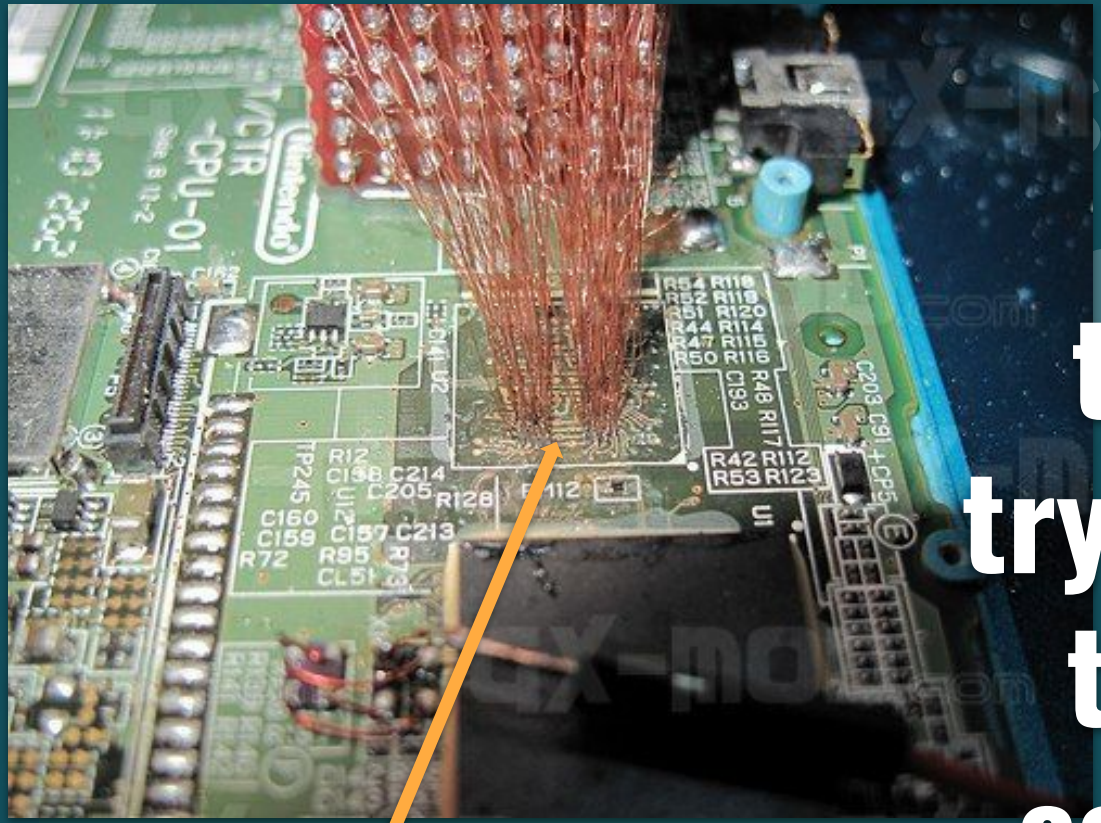
Off-chip memory
some *gigabytes* of storage

On-chip memory
some megabytes of storage

(to scale)



also, who's this Determined Hobbyist?



("neimod" soldered FPGA I/O to this Nintendo 3DS's memory bus)

the person trying to dump their game console RAM

Problems

Confidentiality

“can an attacker read my data?”

Integrity

“can an attacker secretly change my data?”

Today

1. Keystone Framework intro
2. Prior art in Intel SGX
3. Protected paging
4. Evaluation
5. Conclusion

Our Toolset?

Keystone

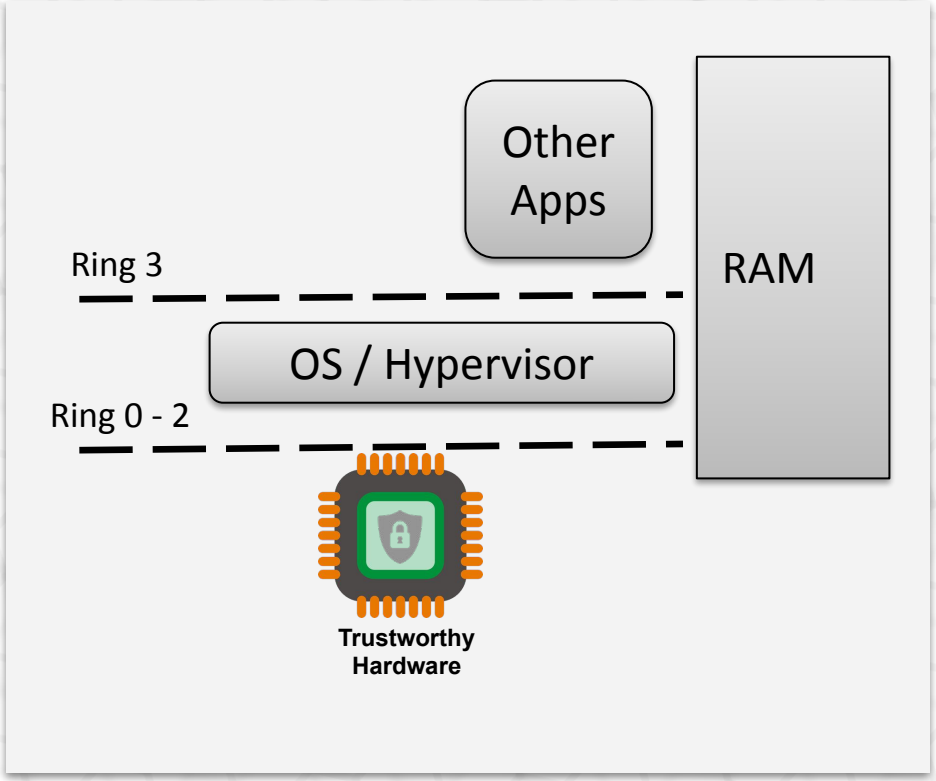
Keystone

an extensible, customizable

Trusted Execution Environment

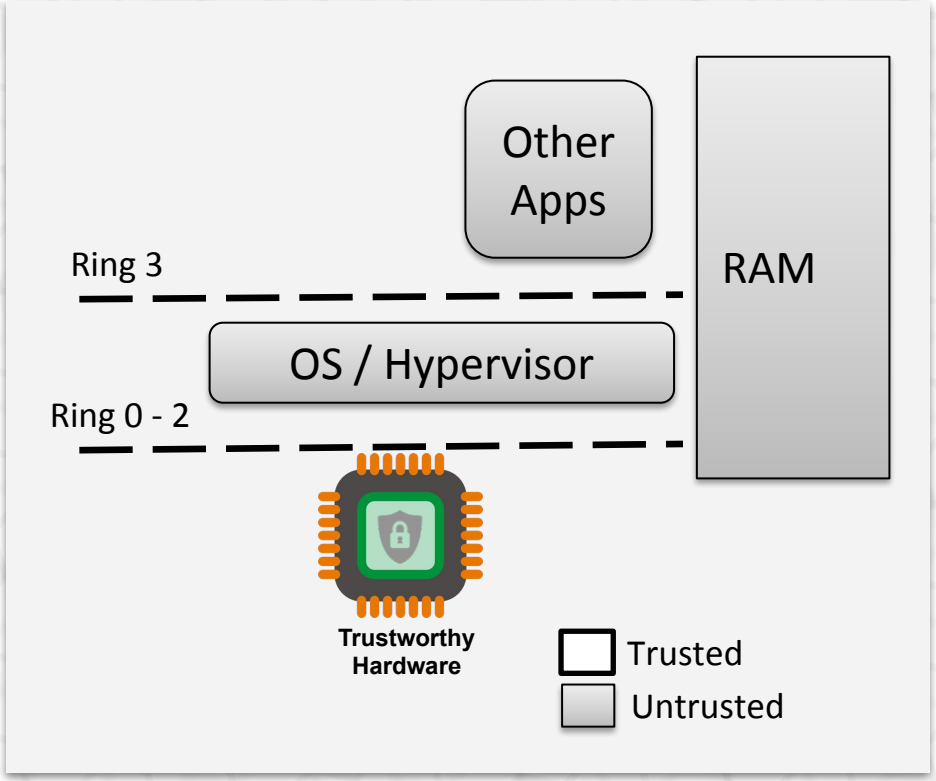
framework for RISC-V

Keystone



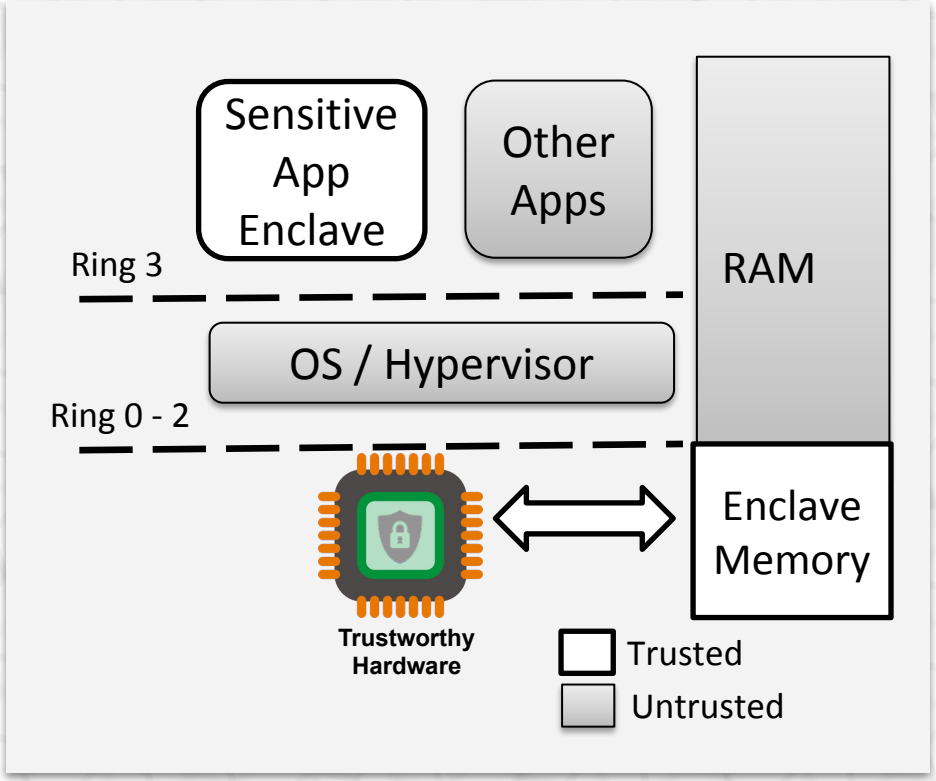
Trusted Execution Environments

Keystone

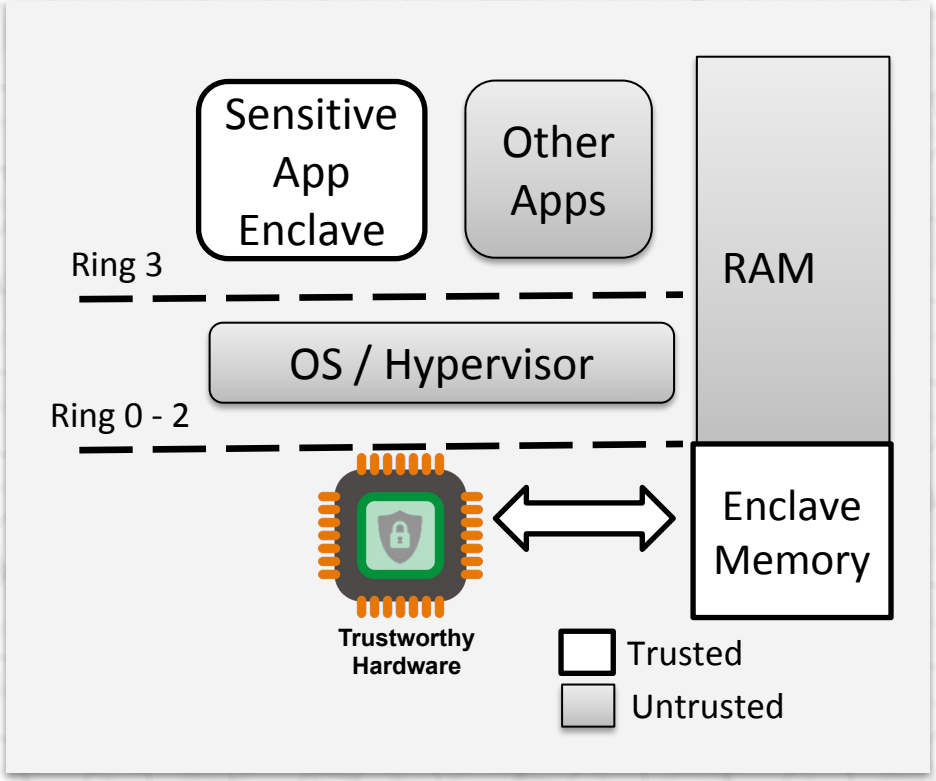


Trusted Execution Environments

Keystone



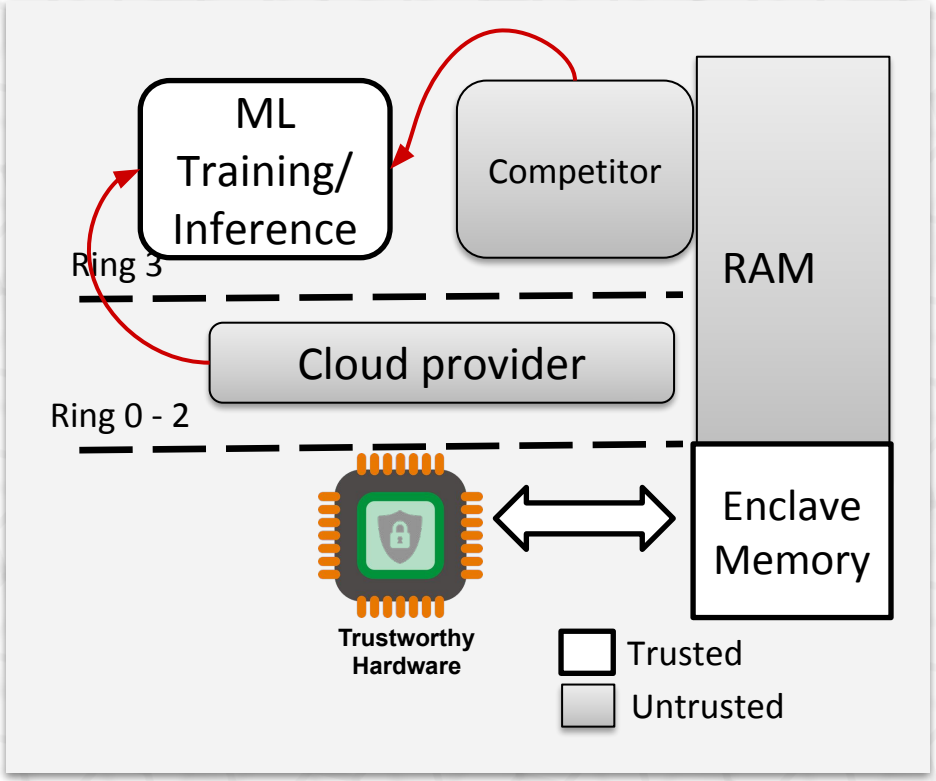
Trusted Execution Environments



Trusted Execution Environments

- Integrity
- Confidentiality
- Remote Attestation

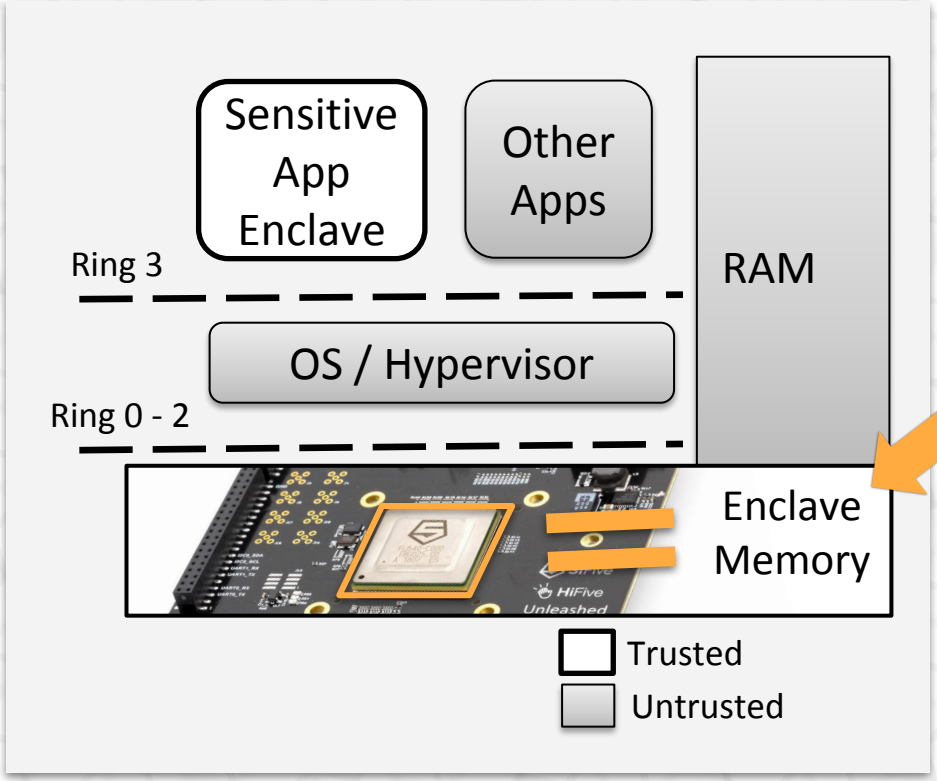
Keystone



Trusted Execution Environments

- Integrity
- Confidentiality
- Remote Attestation

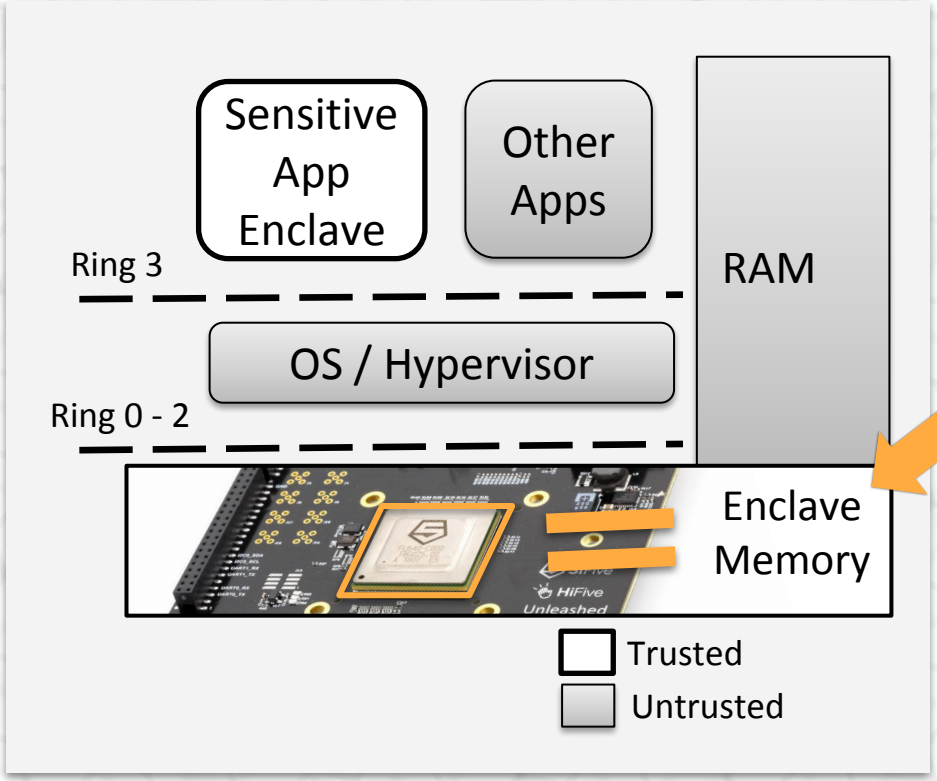
Keystone



Perfect! but...

On-chip memory
some megabytes of storage

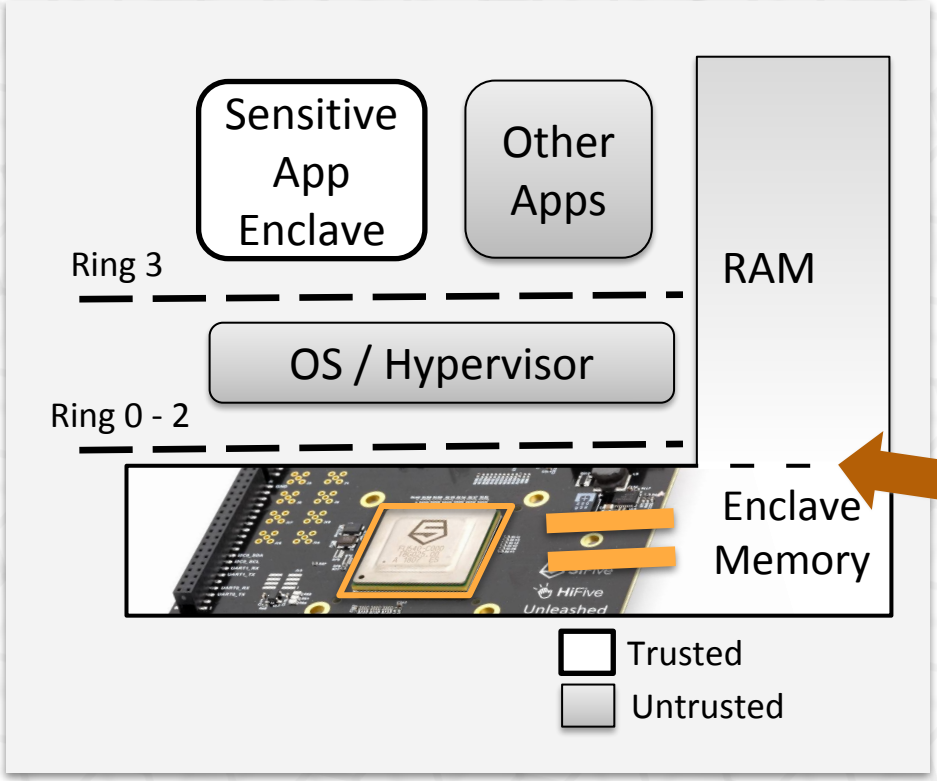
Keystone



Perfect! but...

On-chip memory exhausted very quickly

Keystone



Perfect! but...

On-chi

demand paging

Problems

Confidentiality

“can an attacker read my pages?”

Integrity

“can an attacker secretly change my pages?”

Solutions

Encryption

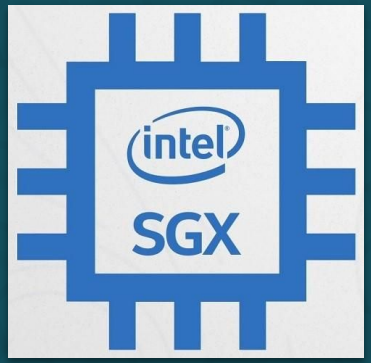
any outbound pages are encrypted

Hashing

any inbound pages have their hashes checked

Precedent

Confidentiality **Integrity**



Intel’s Secure Guard Extensions solve in hardware

Precedent

Confidentiality

Modified AES-CTR, 128-bit

Version counters for replay protection

512b block granularity

Integrity

Carter-Wegman MAC, 56-bit

Merkle tree hash storage

A Software Approach

for commodity RISC-V hardware

Confidentiality

AES-CTR, 256-bit

Version counters for replay protection

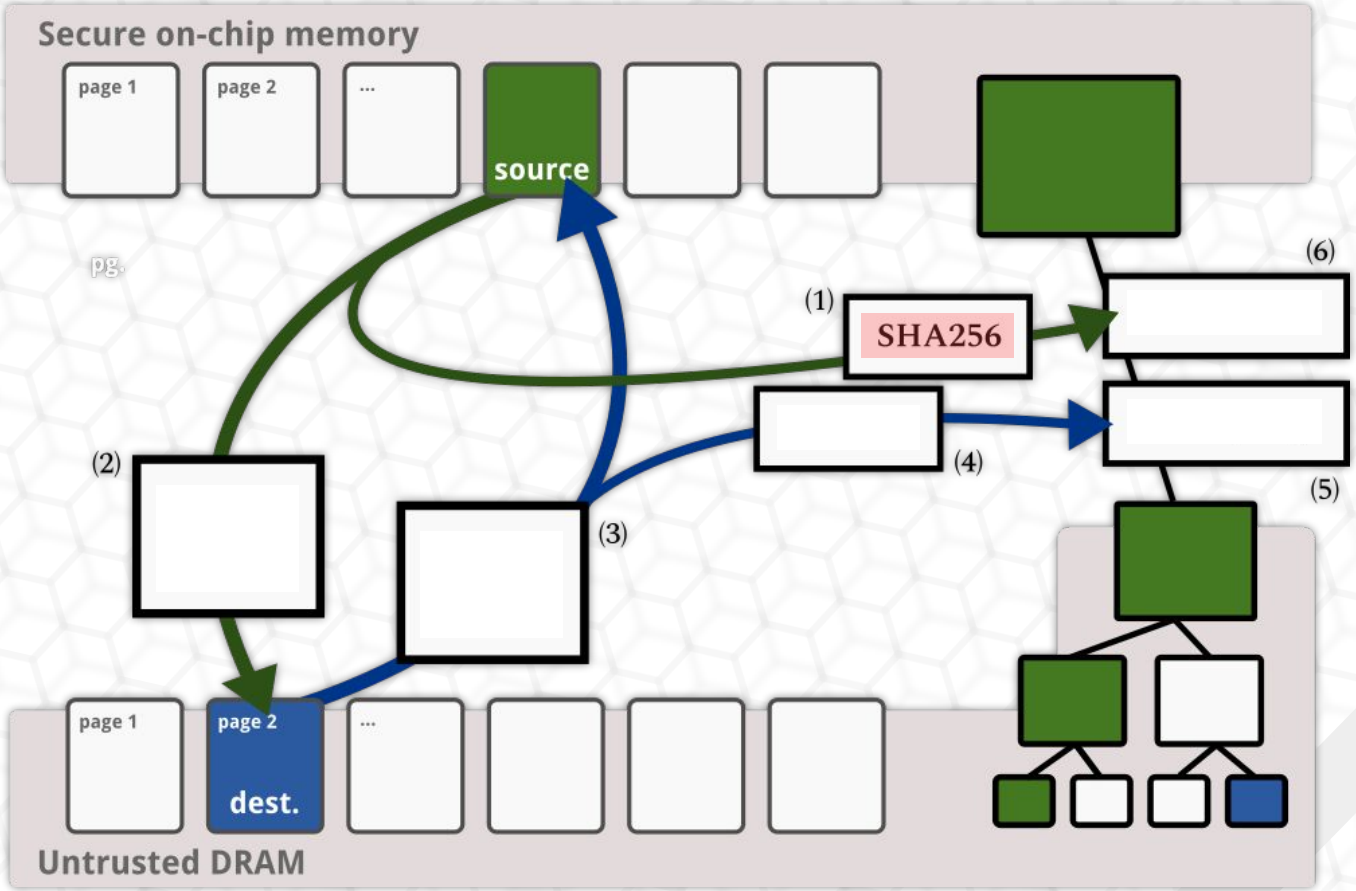
page size granularity (multiple of 4096b)

Integrity

SHA256

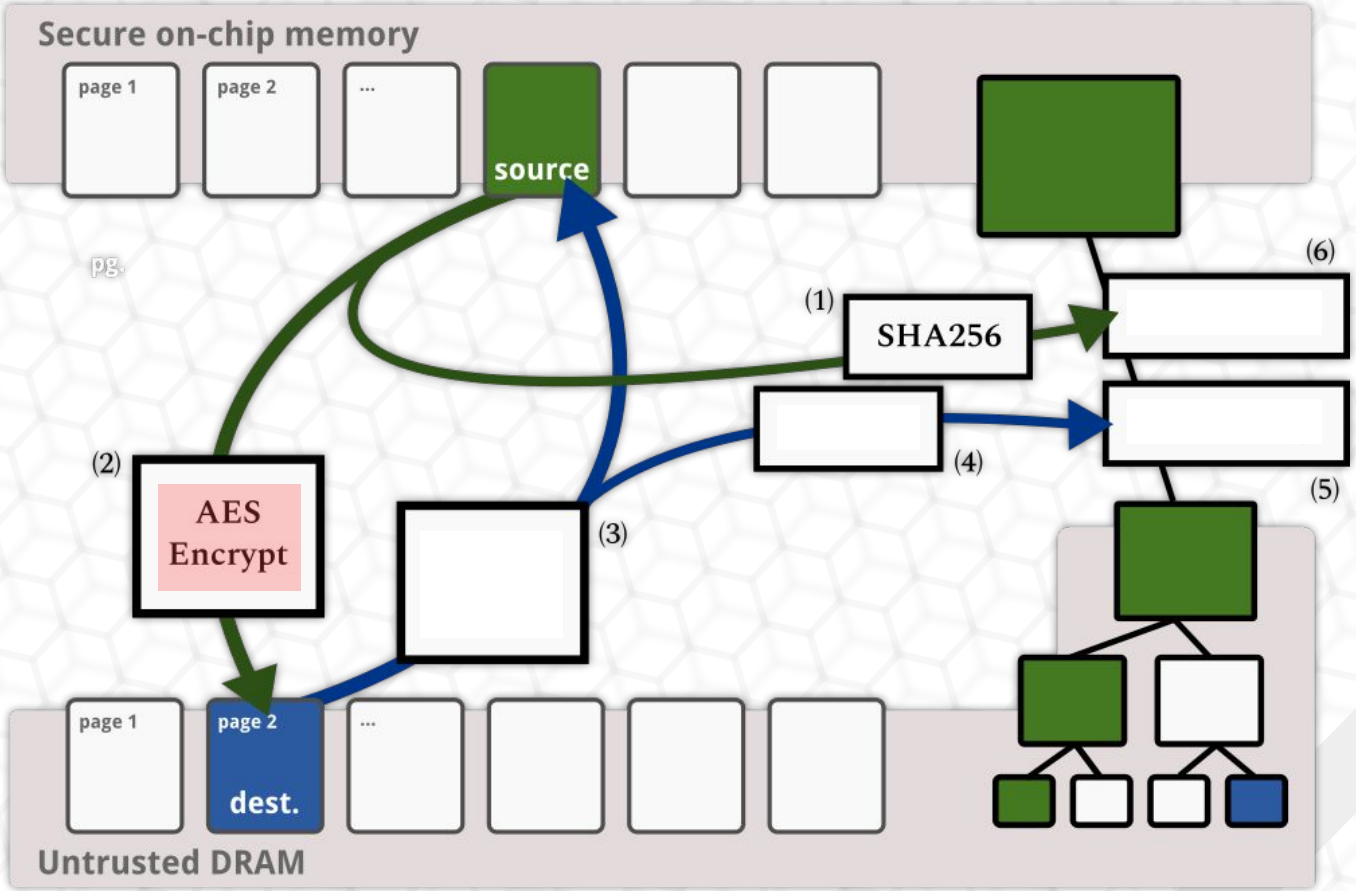
Merkle tree hash storage

The Scheme



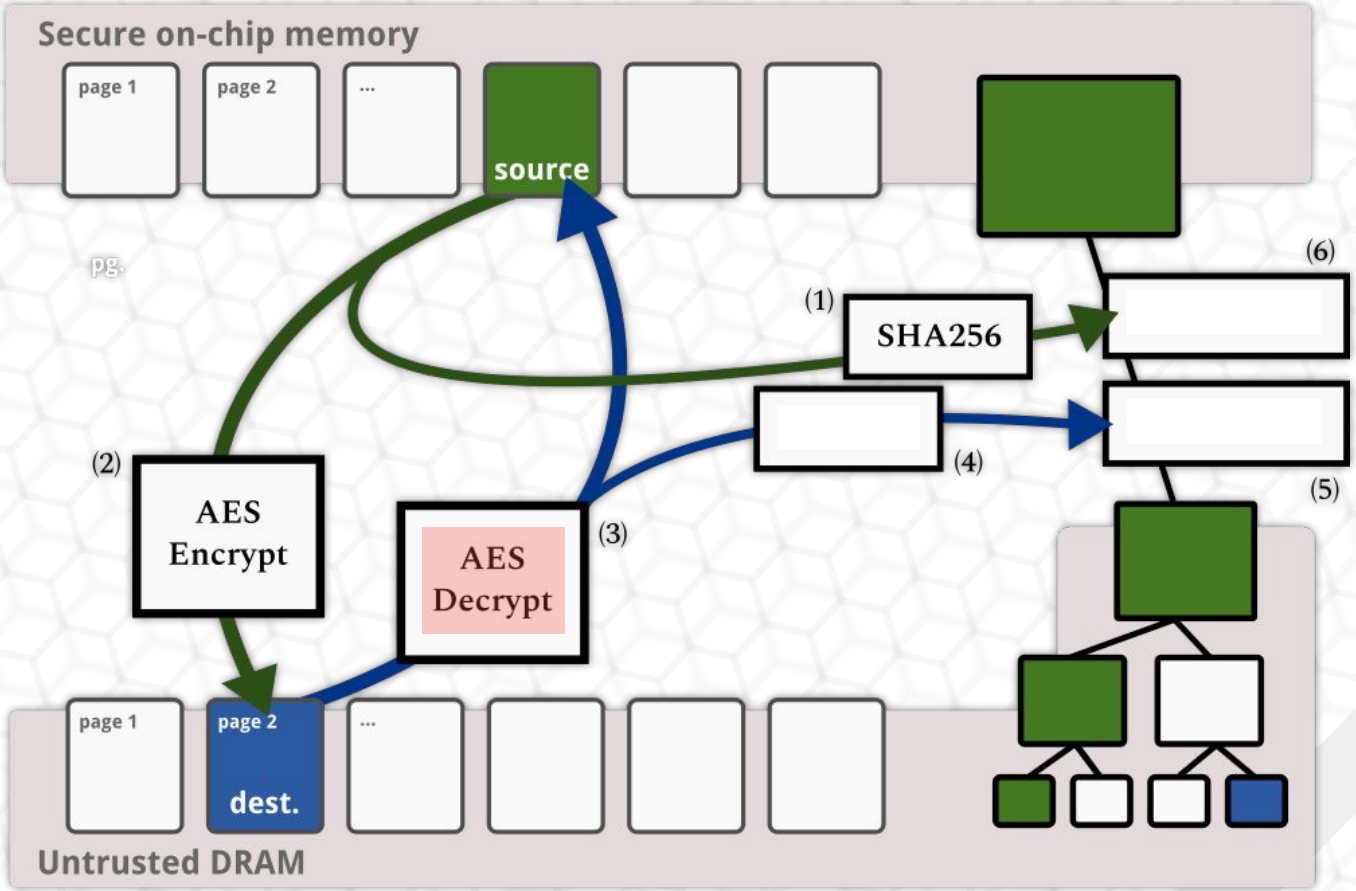

```
s_hash := sha(s)
```

The Scheme



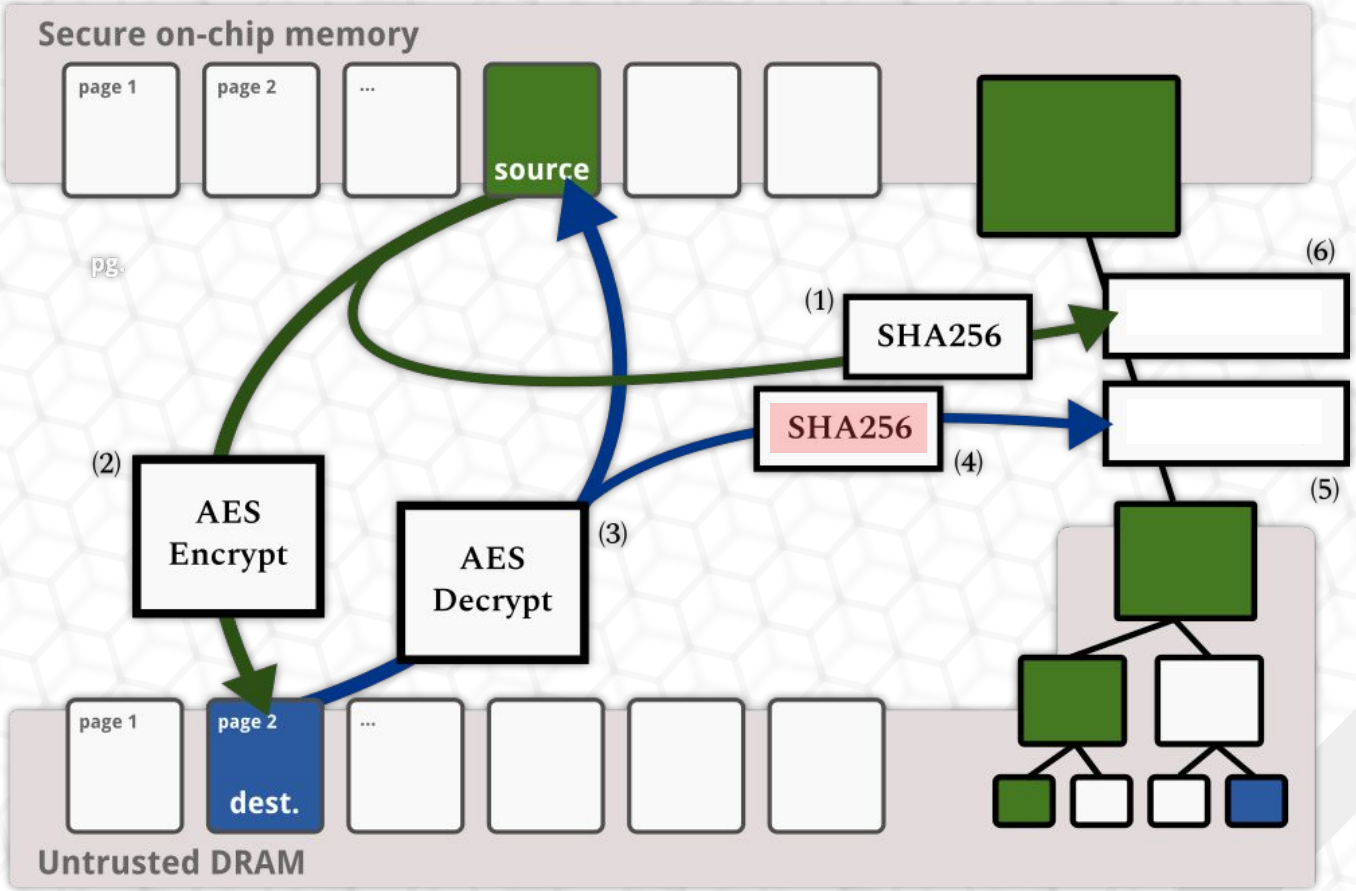

```
s_hash := sha(s)
s_enc := aes(s)
```

The Scheme



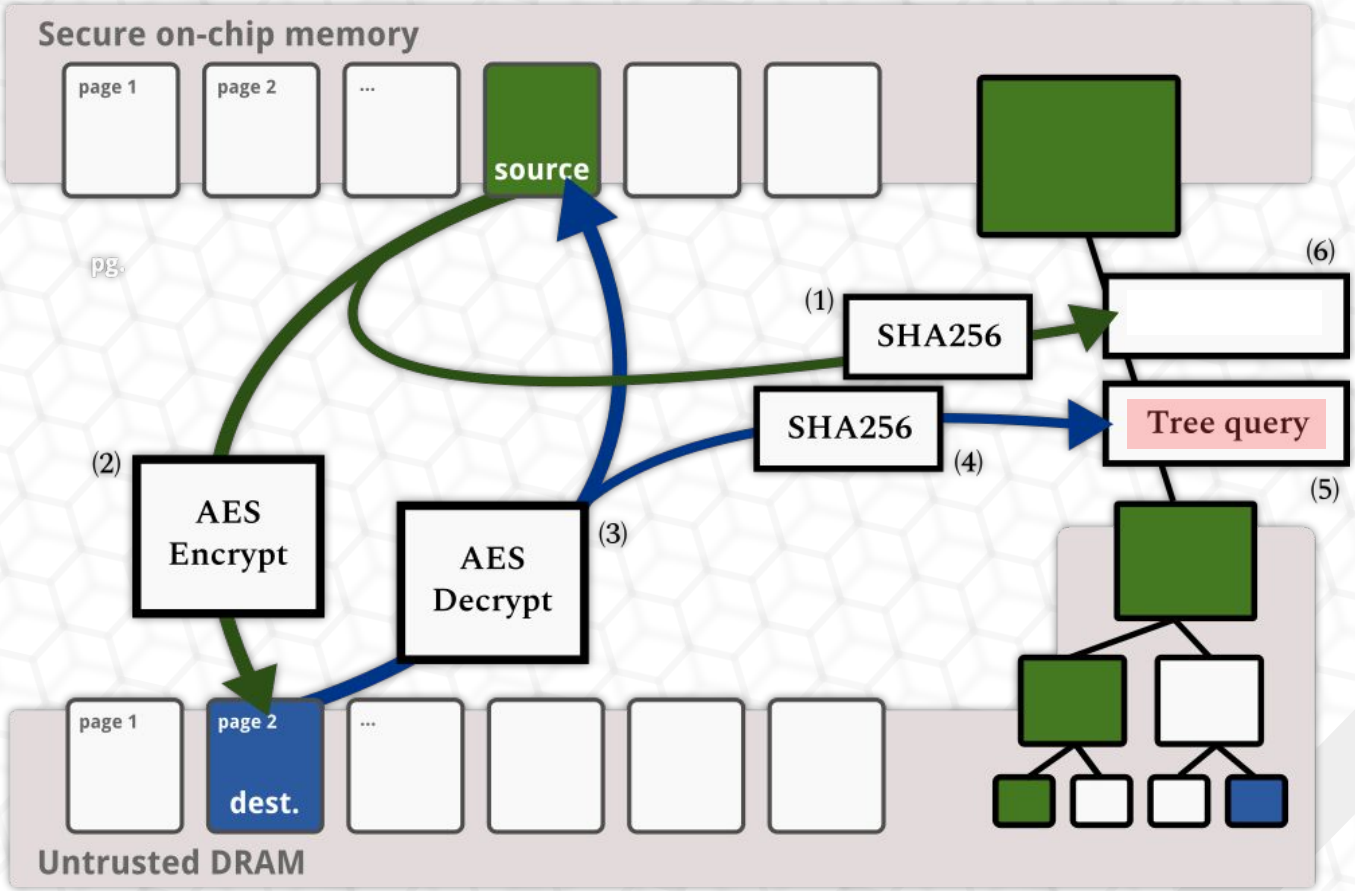

```
s_hash := sha(s)
s_enc := aes(s)

d_dec := aes(d)
```

The Scheme



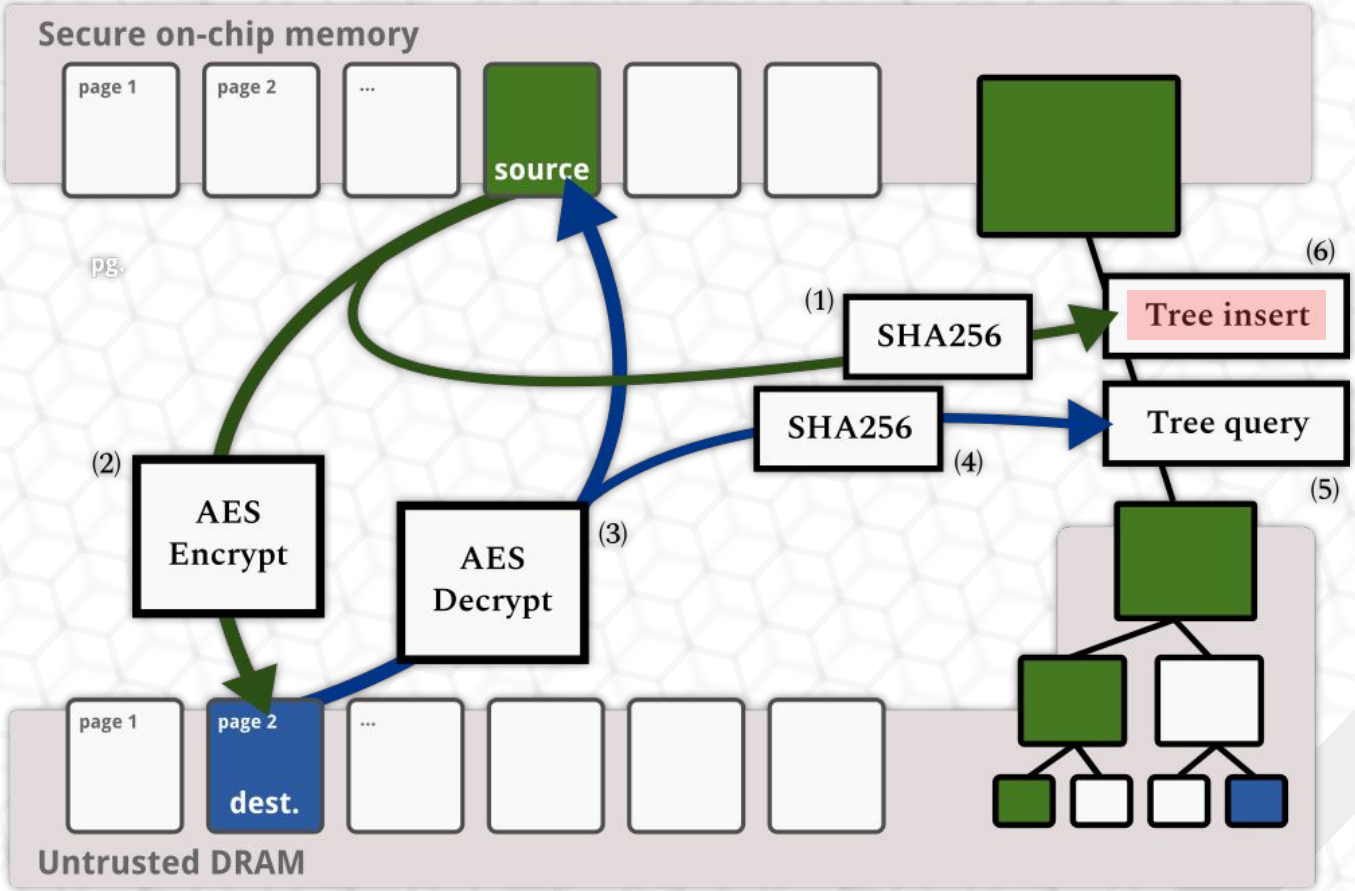

```
s_hash := sha(s)
s_enc := aes(s)

d_dec := aes(d)
d_hash :=
    aes(d_dec)
```

The Scheme




```
s_hash := sha(s)
s_enc := aes(s)

d_dec := aes(d)
d_hash :=
    aes(d_dec)

check_hash(d_hash)
```

The Scheme



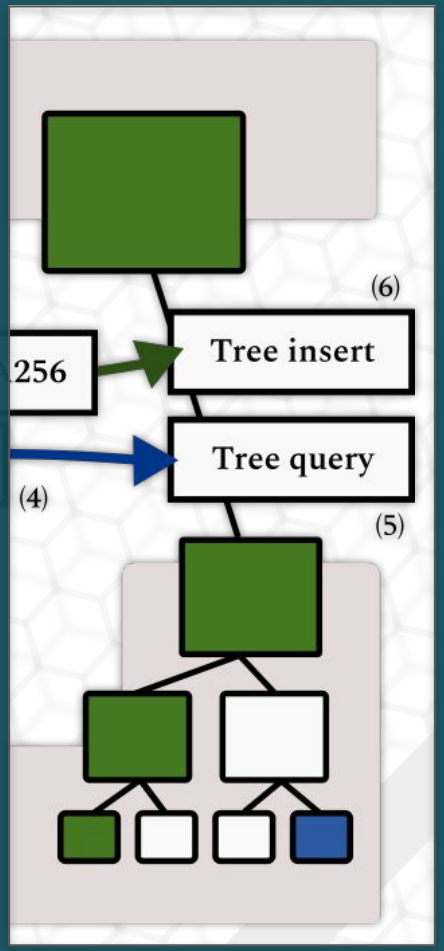

```
s_hash := sha(s)
s_enc := aes(s)

d_dec := aes(d)
d_hash :=
    aes(d_dec)

check_hash(d_hash)
store_hash(s_hash)
```

Why a tree?

Array of page hashes too big (wasteful) for on-chip memory

↓

Move it *off-chip*

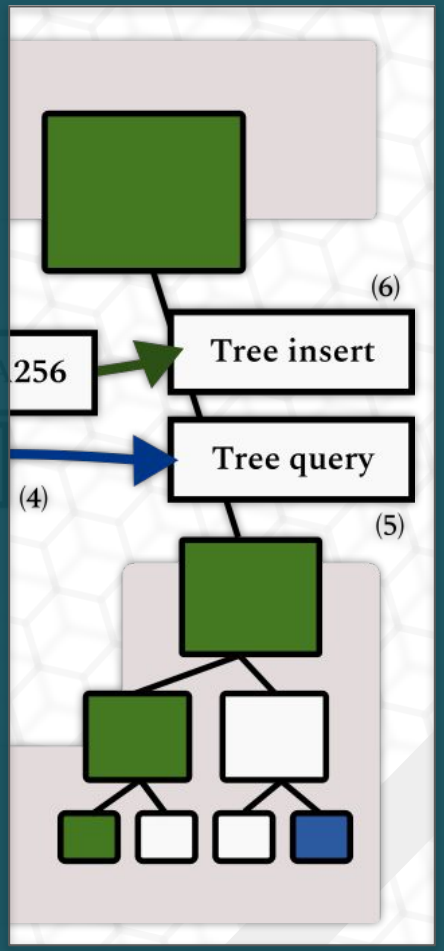


Why a tree?

Off-chip hashes

↓

<u>Untrusted</u> hashes?



Problems

Confidentiality

“can an attacker read my hashes?”

Integrity

“can an attacker secretly change my hashes?”

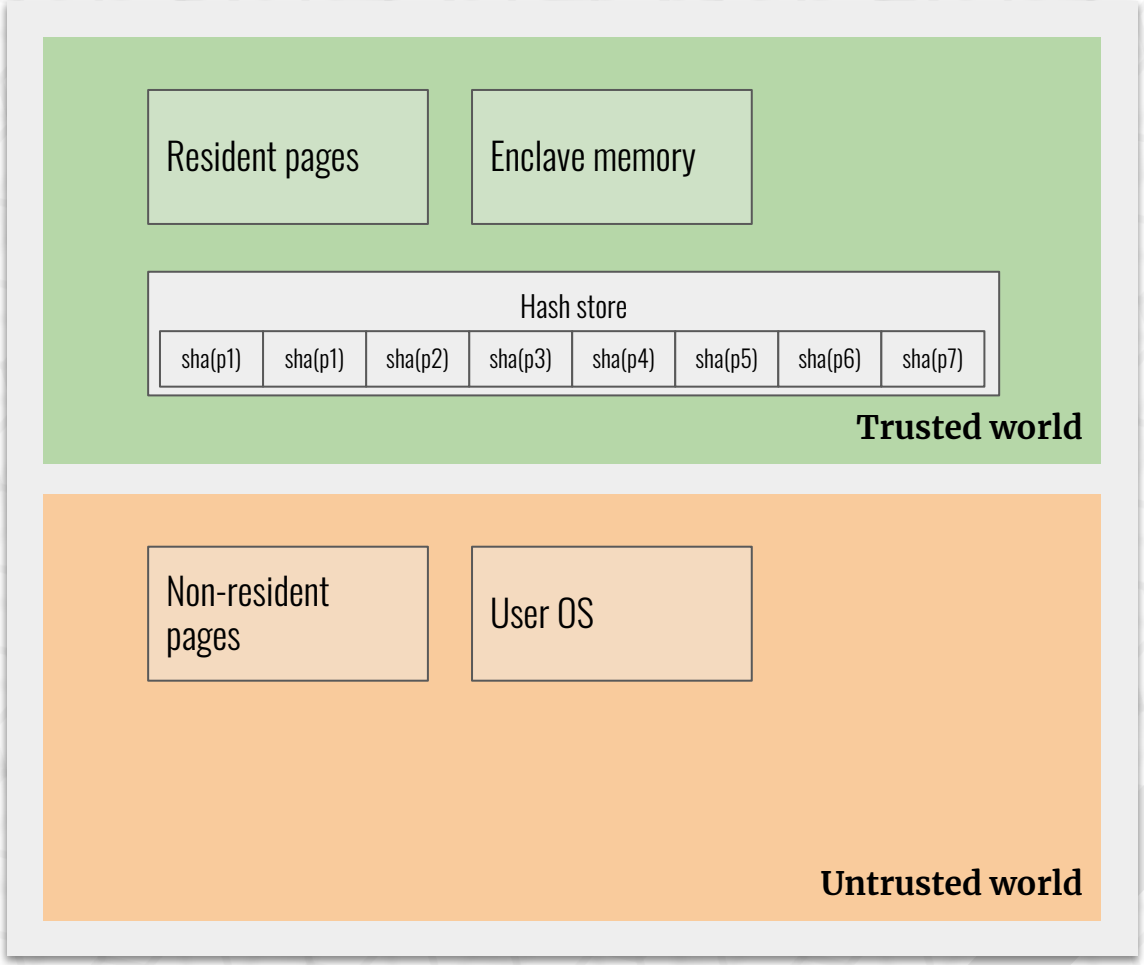
Solutions

Don’t care!

cryptographic hashes leak no information

More hashing!

hash the hash store, keep the root hash safe



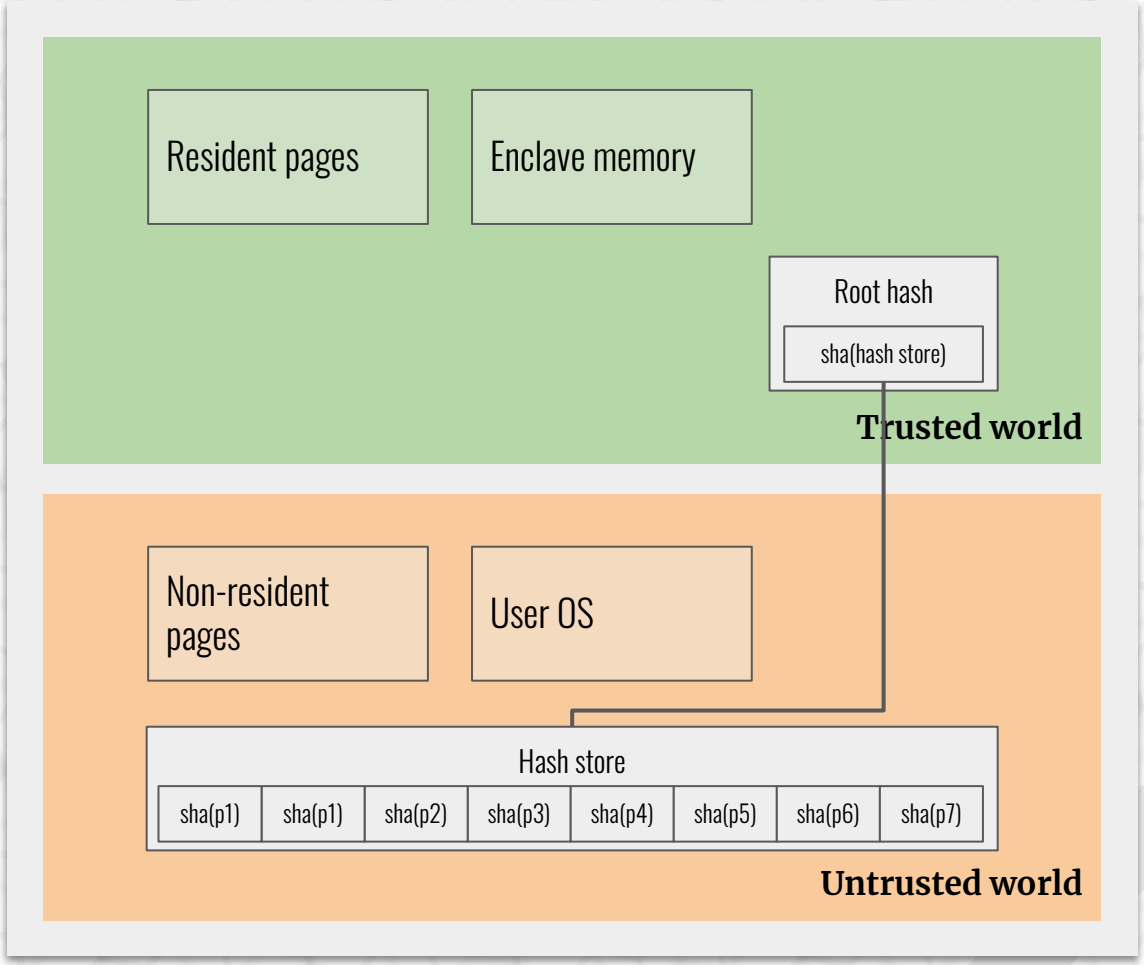
Hashing the store

Original plan:

Keep nonresident page hashes in secure memory

Problem:

On-chip memory too valuable!



Hashing the store

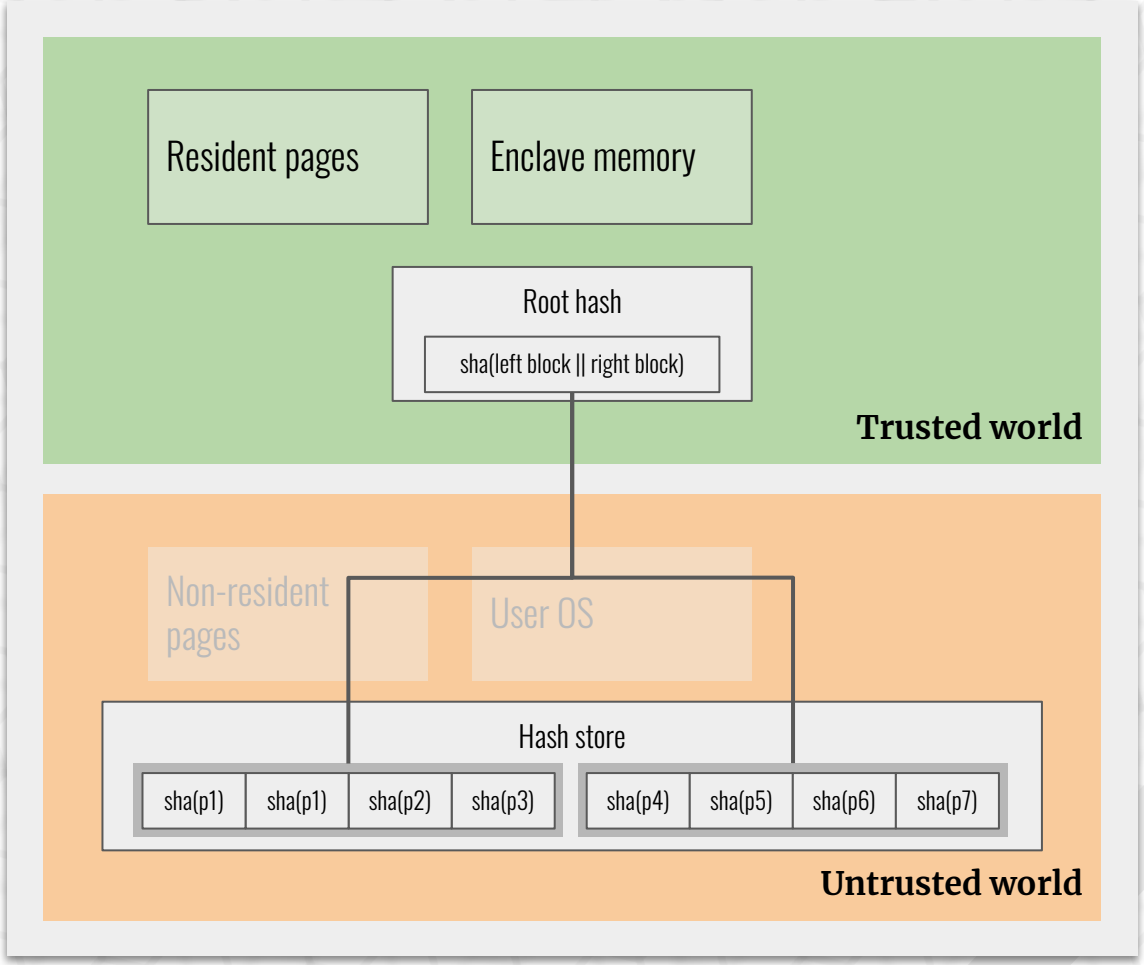
Solution:

Move store off-chip, check its integrity during page swaps

Problem:

Hashing the entire store wastes CPU cycles

Hashing the store



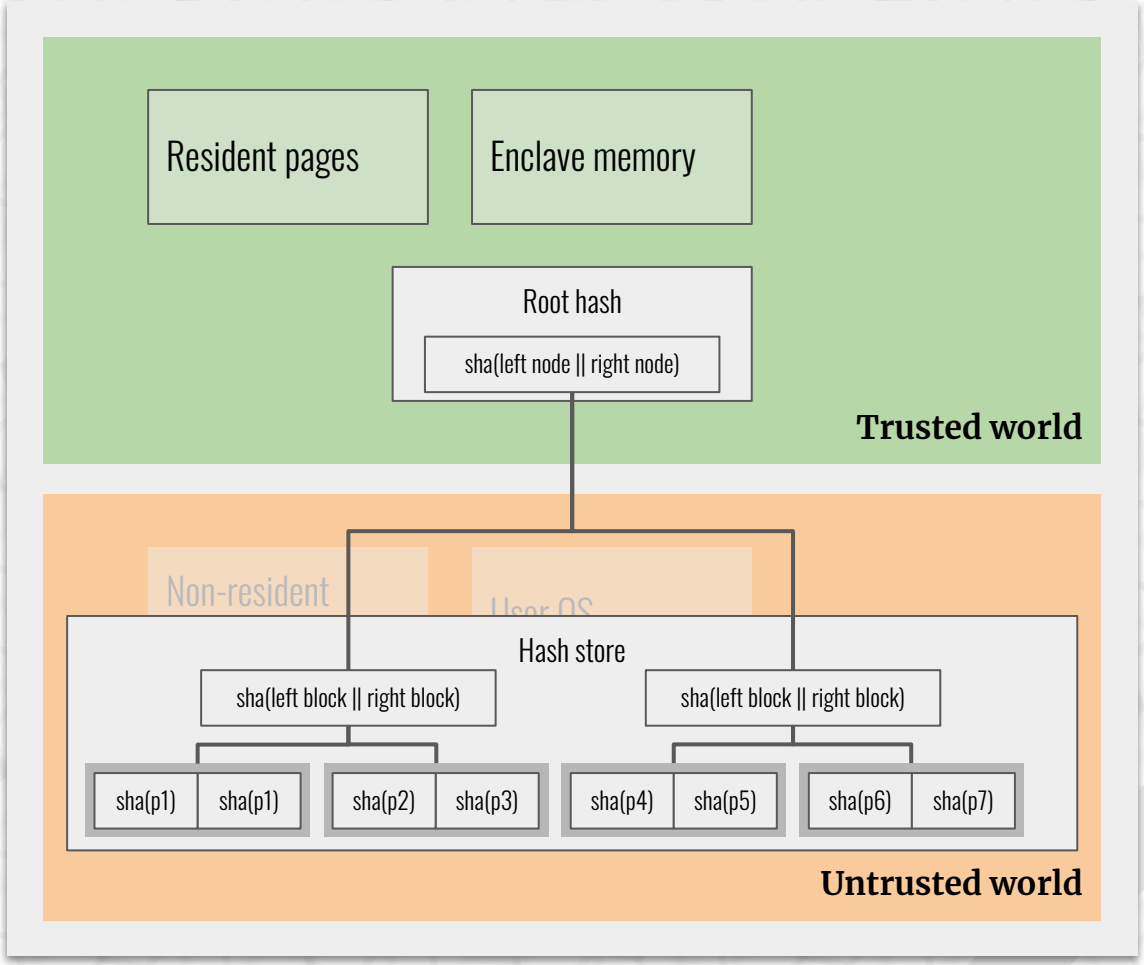
Solution:

- Split hash store
- Hash left or right side as needed
- Propagate to root

Problem:

Hashing half of store still too much for one page swap

Hashing the store

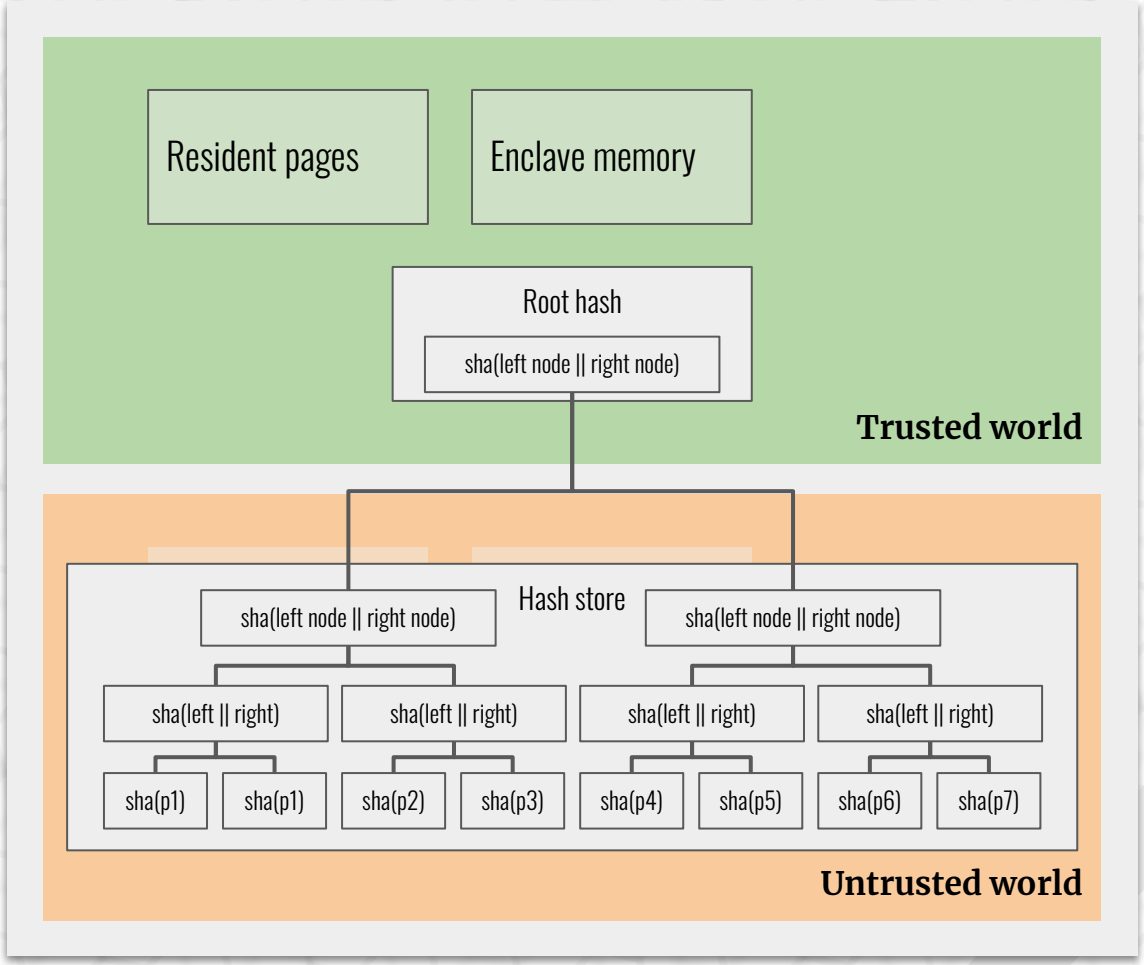


Solution:

- Recursive splits
- Hash only the relevant leaf
- Propagate to root

This tree structure is called a *Merkle Tree*.

Hashing the store



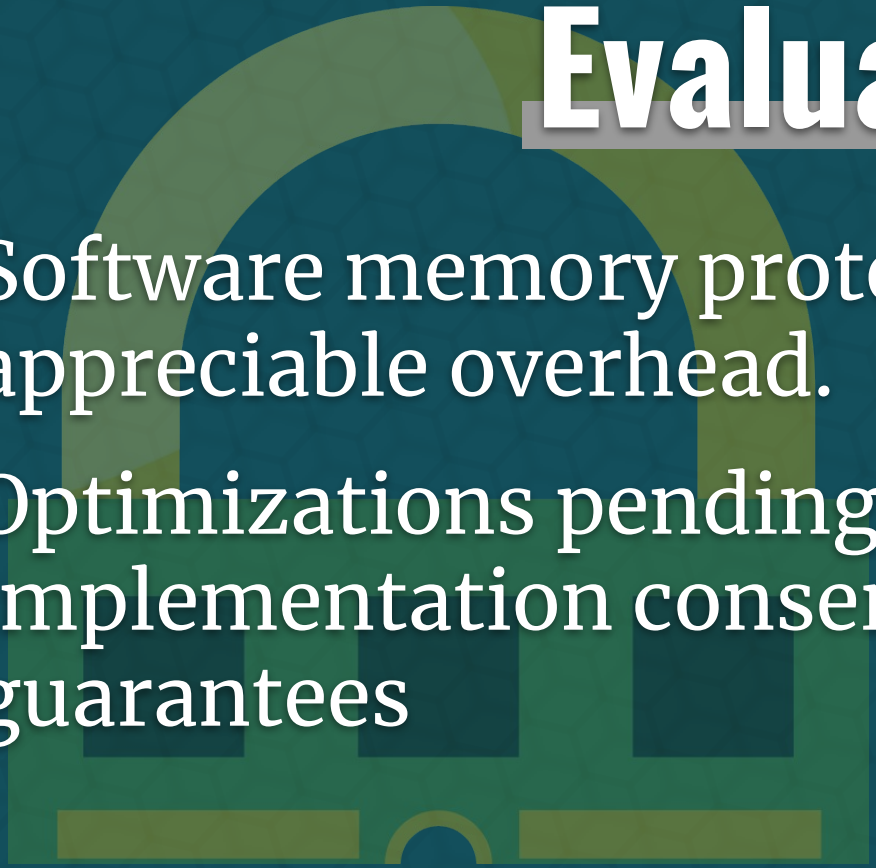
Merkle tree tradeoffs:

- Very little secure memory usage
- Deeper tree needs more insecure mem
- Deeper tree hashes fewer bytes total
- Deeper tree needs more hash passes

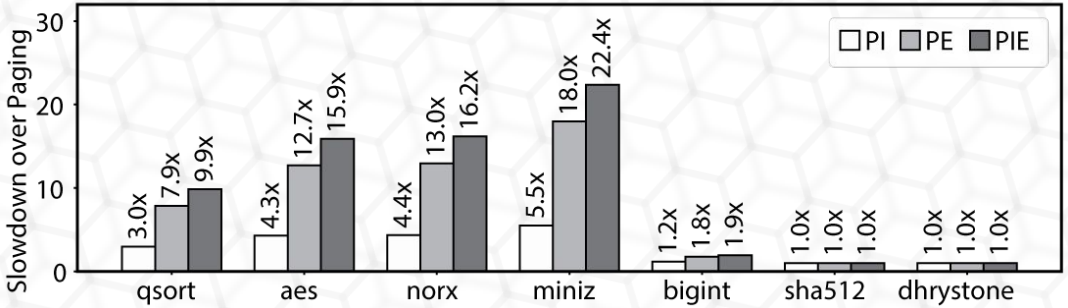
Evaluation

Software memory protection feasible, with appreciable overhead.

Optimizations pending; current implementation conservative with security guarantees

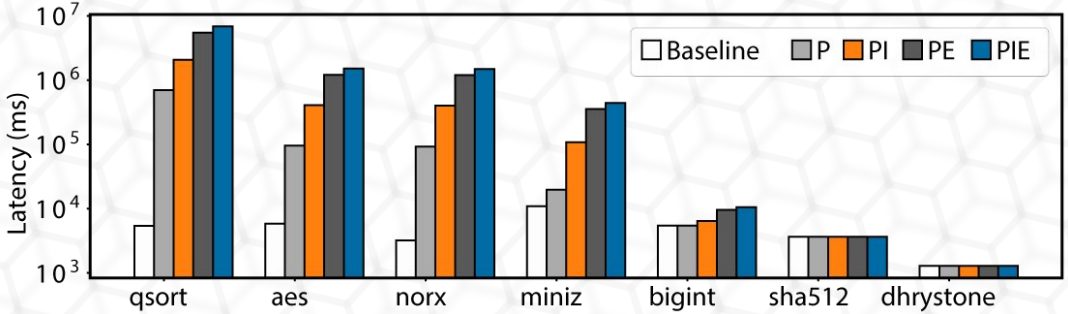


Evaluation



With efficient paging infrastructure, even unoptimized protection routines could be viable.

Evaluation

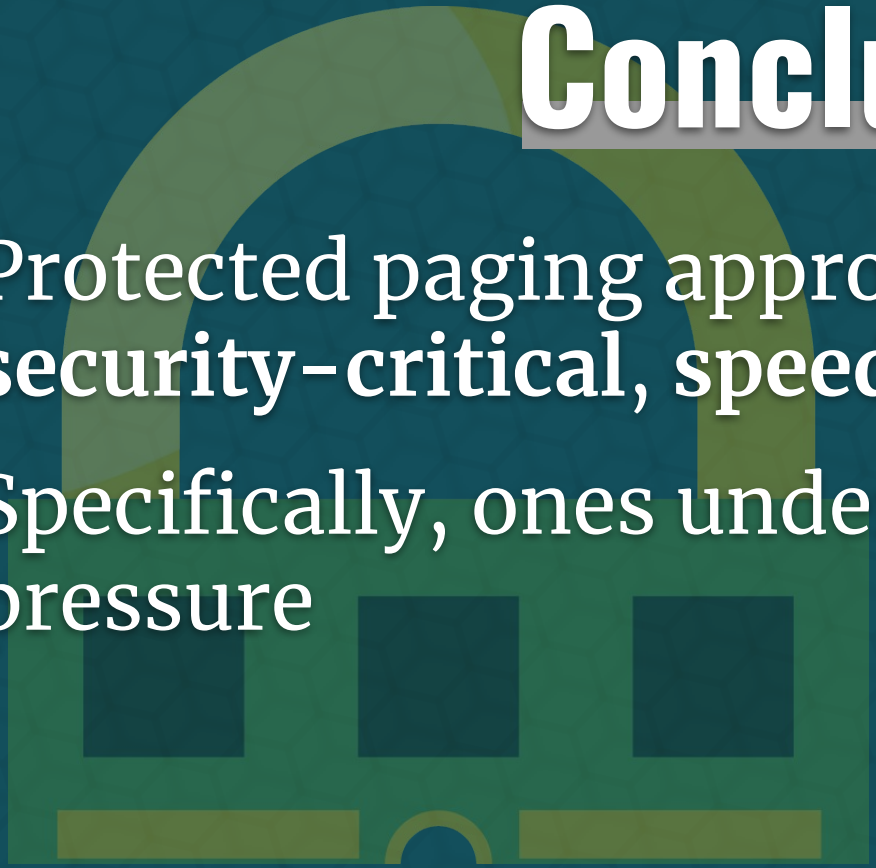


Unfortunately, paging currently accounts for huge runtime overheads

Conclusion

Protected paging appropriate for **security-critical**, **speed-flexible** operations

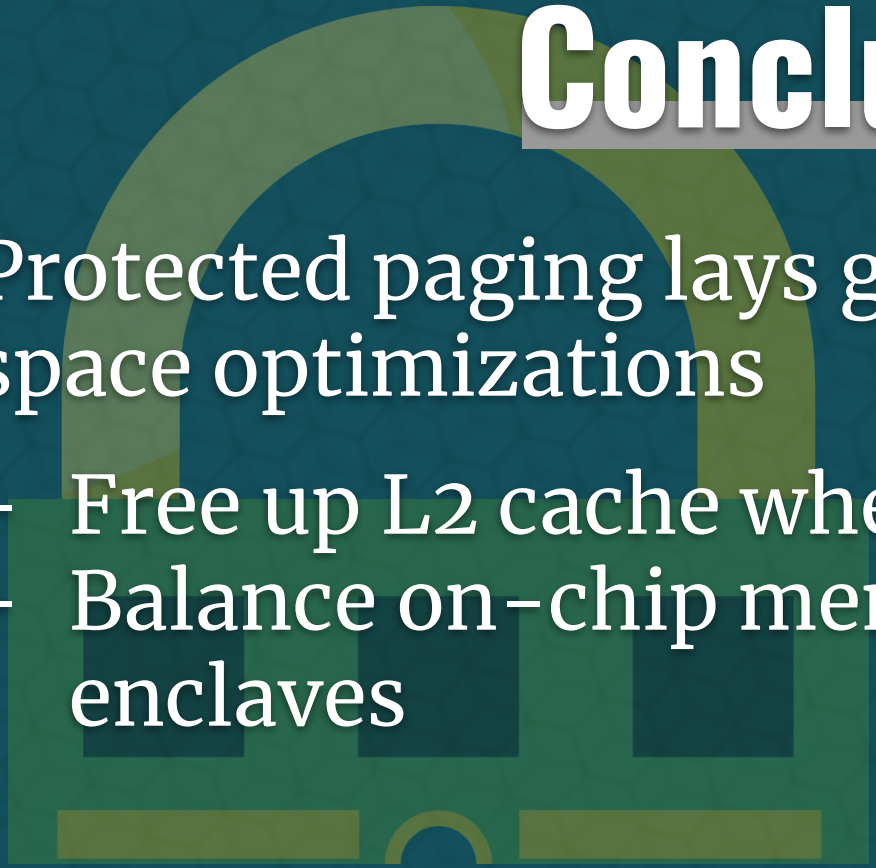
Specifically, ones under secure memory pressure

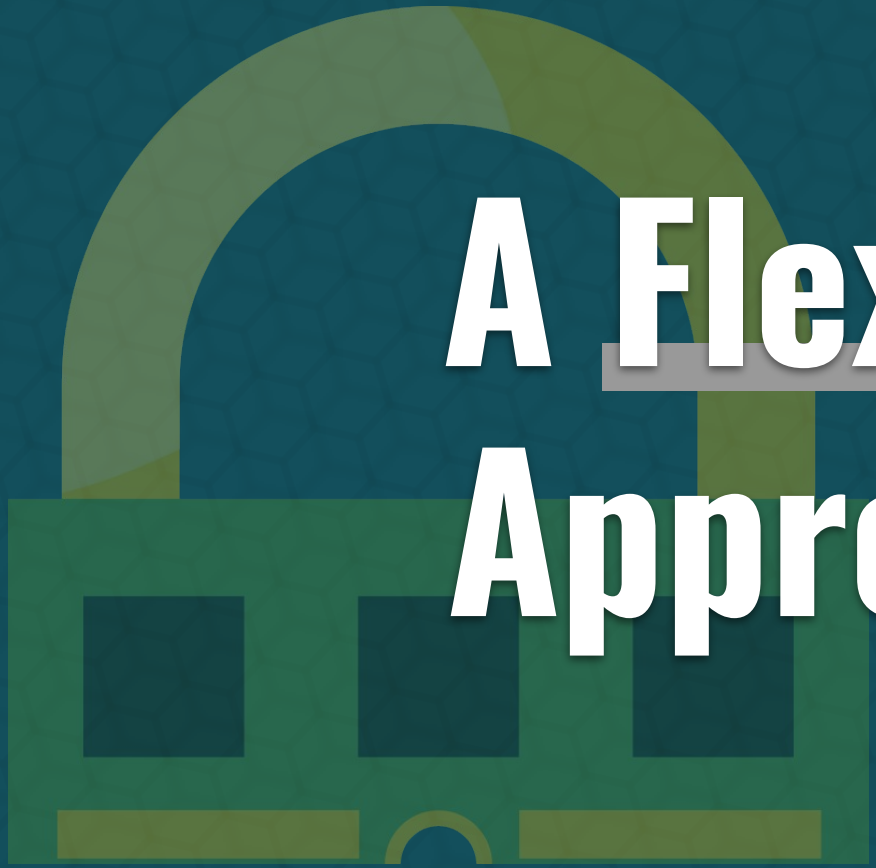


Conclusion

Protected paging lays groundwork for other space optimizations

- Free up L2 cache when enclave is idle
- Balance on-chip memory among several enclaves



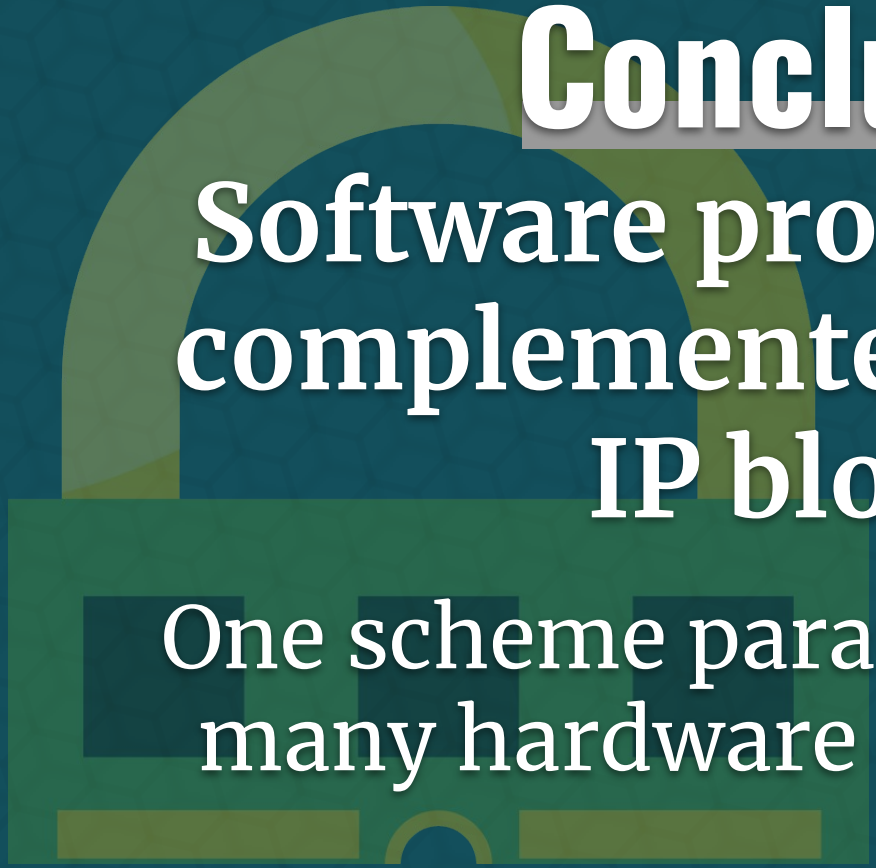


A Flexible Approach

Conclusion

Software protections need no special IP blocks!

Frees up die area for cost constrained hardware



Conclusion

Software protections are complemented by special IP blocks!

One scheme parametrizable over many hardware configurations