Does Near Data Processing Have Potential to Improve Secure Memory Overheads?

Casey Nelson[1], R. Iris Bahar[1], and Tamara Lehman[2]

Brown University[1], University of Colorado Boulder[2]

Goals

1. Develop a scheme for fetching security metadata using NDP
2. Determine the potential for this scheme by running approximation simulations

Overview

1. Background
2. Proposed Design
3. Implementation
4. Evaluation
5. Conclusion

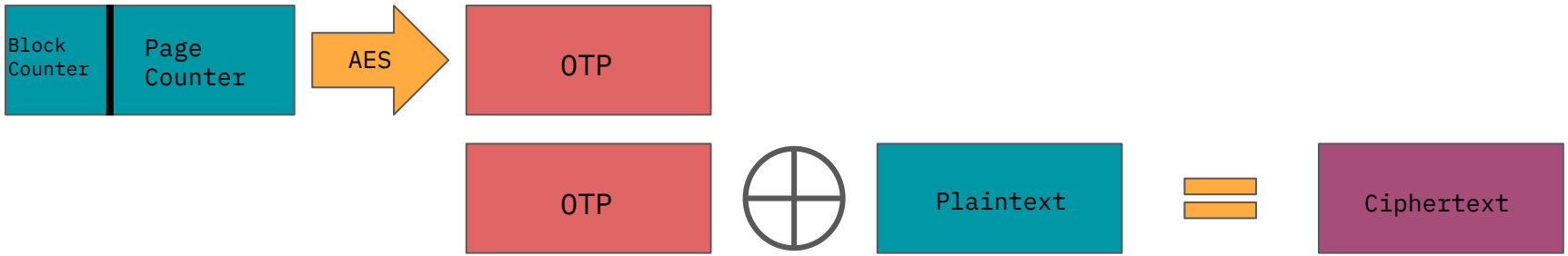
Background

Background: Security Metadata

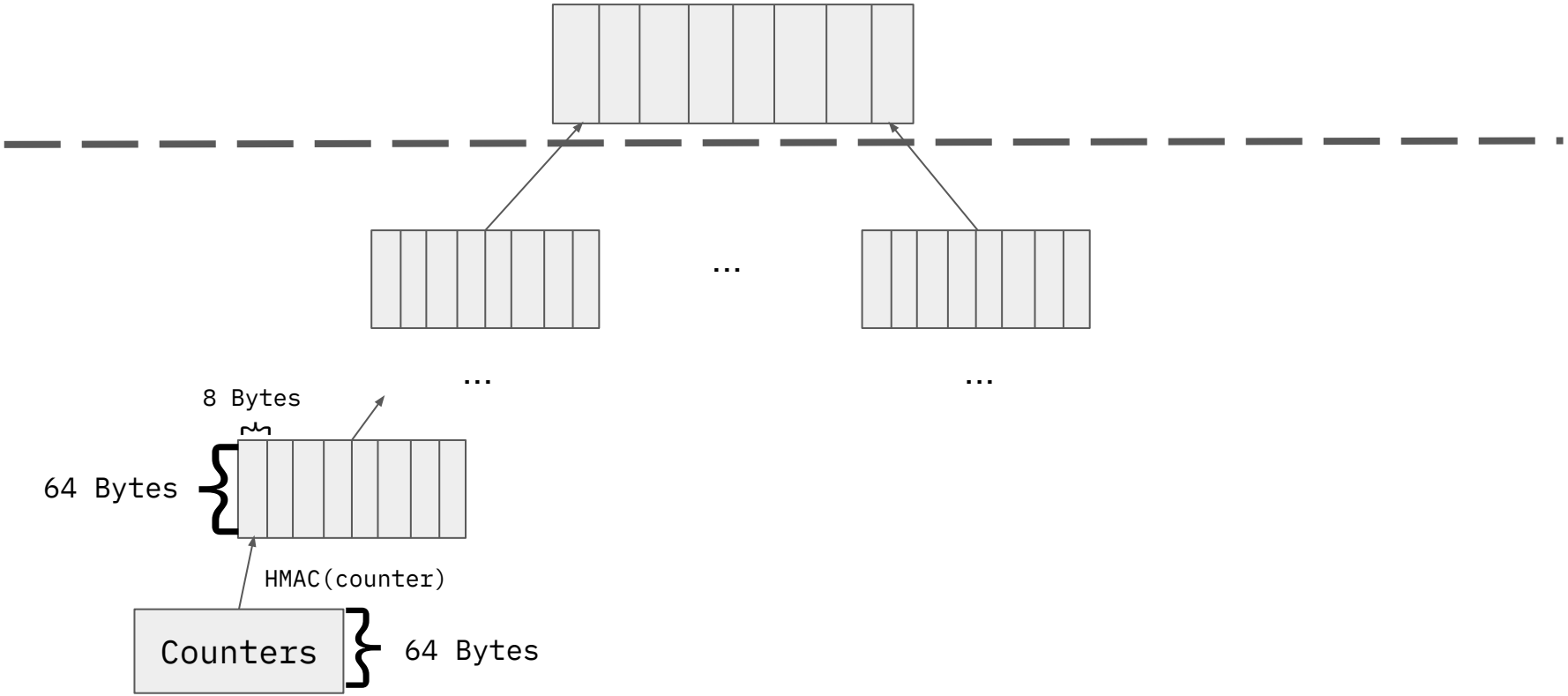
1. Counters
2. Tree Nodes
3. Data HMACs

Counters

- Block Counters
  - 7 bit per 64B data block
  - Updated every time it's used to encrypt data
- Page Counters
  - 8B per 4kB page
  - Updated every time a block counter on this page overflows
  - Requires re-encrypting entire page (every block)
- Both counter used to encrypt data:



Bonsai Merkle Tree Nodes



Data HMACs

- HMACs = keyed hashes of the ciphertext || counter
- Ensures the integrity of accessed data
  - Counter verified with BMT
  - Ciphertext || verified counter
  - HMAC(Ciphertext, Counter) != HMAC from memory → tampered data

Metadata Caching

- Metadata can be safely cached since the cache is located on chip (in the trusted computing base)
  - Close proximity of cache to CPU
  - Not accounting for side channels
- Metadata Cache hits greatly reduce overhead
  - Counter hit means no tree nodes need to be fetched
  - A tree node hit means you only need to fetch a fraction of the tree levels
- Metadata can be cached in:
  - LLC
  - Dedicated in the metadata cache

The Problem with Metadata Caching

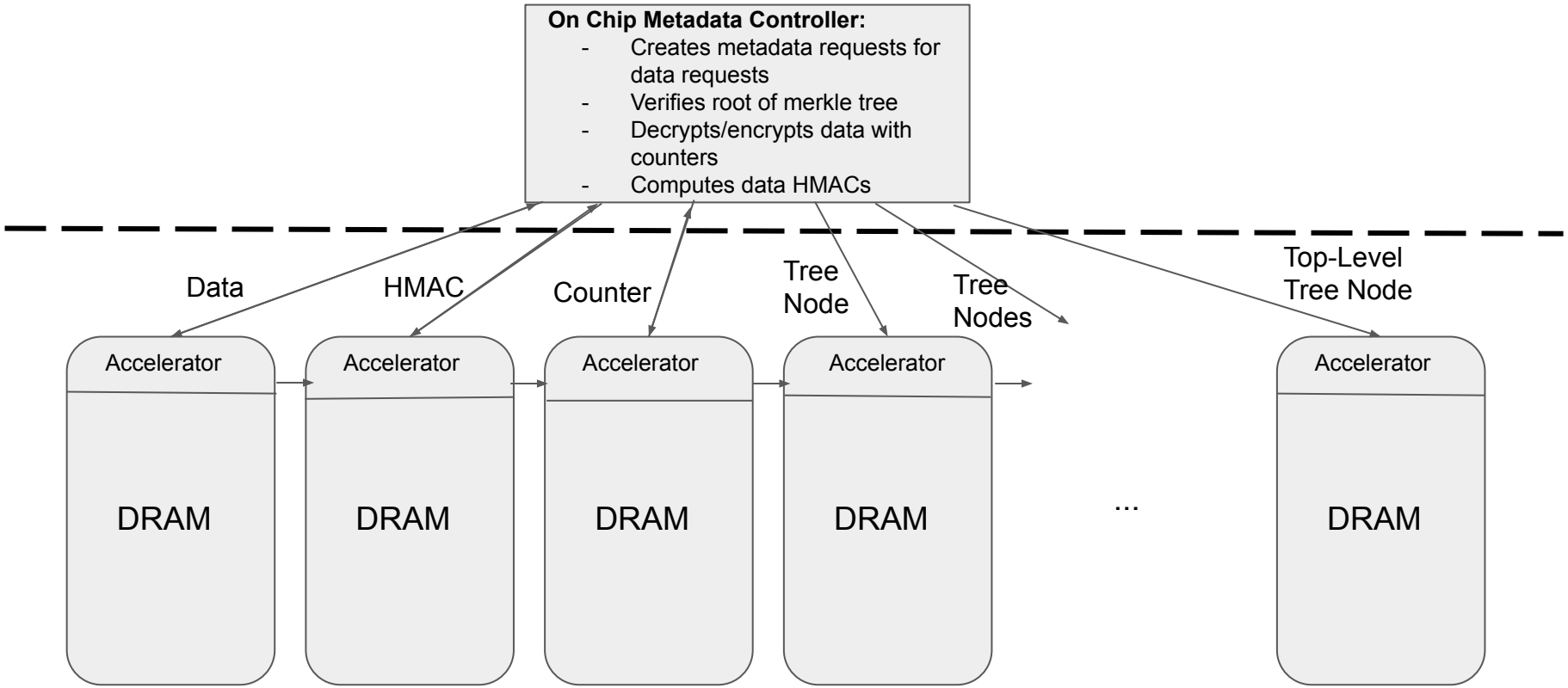
- Caching in the LLC
  - Causes contention for space with regular data
- Dedicated metadata cache
  - Different metadata types of conflicting reuse distance and miss costs
    - Counters:
      - High miss cost
      - Long reuse distance
    - Tree nodes
      - Miss costs decrease as reuse distance shortens
  - Metadata blocks are interdependent

Enter: Near Data Processing

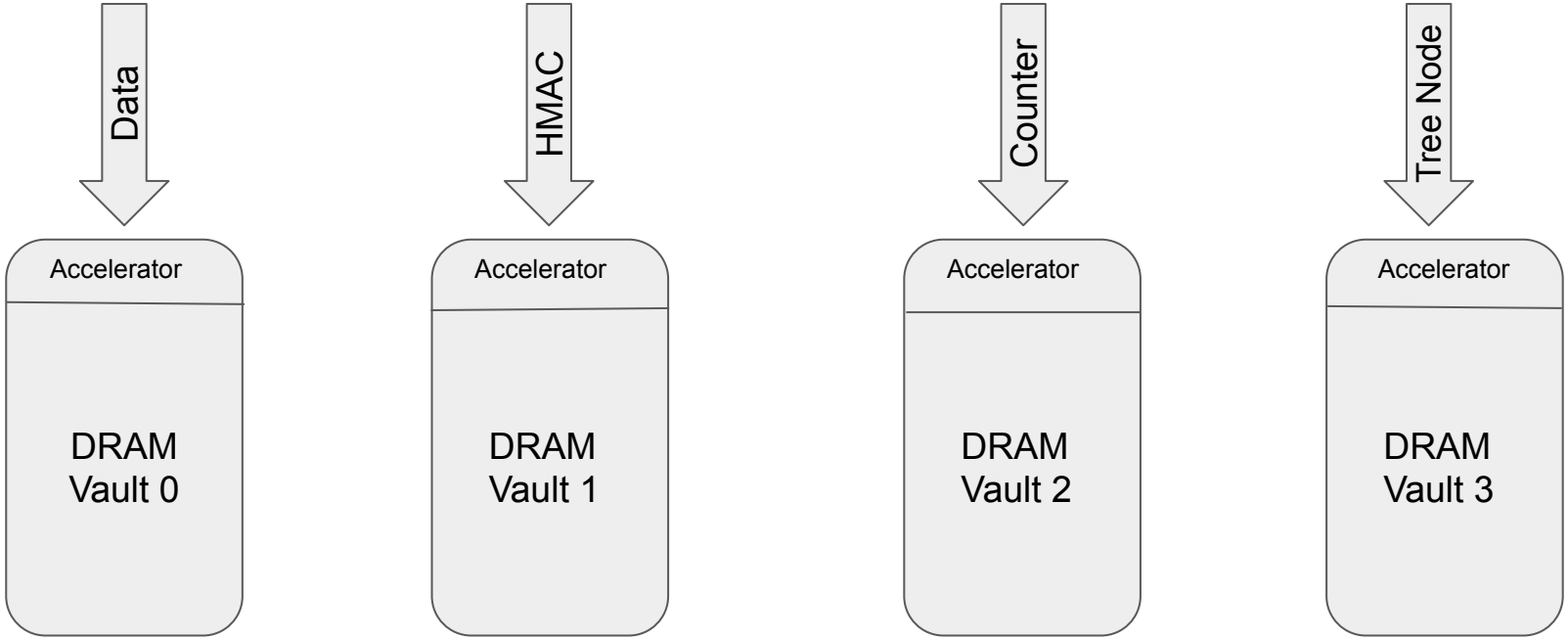
- Near data processing can be used to alleviate metadata cache contention
- Eliminate metadata requests from the critical path by accessing them in parallel
  - As opposed to eliminating them from the critical path by reducing accesses
- Take advantage of the architecture
  - 16 DRAM vaults = up to 16 parallel metadata requests and hash computations

Design

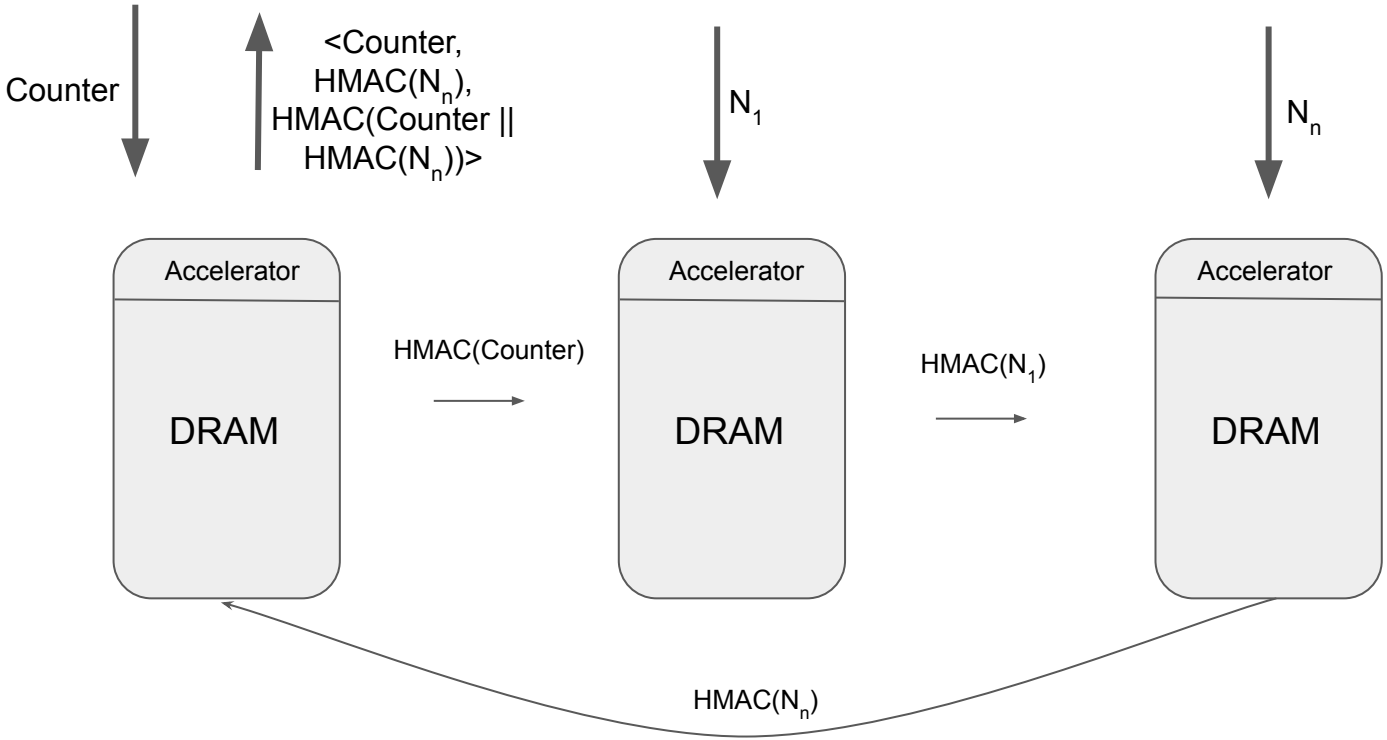
Proposed Scheme



Step 1:
Create Metadata Requests and Forward to Memory



Step 2:
Near Data Accelerators Handle Requests



Step 3:
Metadata Controller Verifies Results

- <Counter, HMAC(N_n), HMAC(Counter || HMAC(N_n))>
  - Metadata controller computes H = HMAC(Counter || HMAC(N_n))
  - Assert H = HMAC(Counter || HMAC(N_n)) returned
  - Needed to prevent replay attack
    - Key: Attacker cannot compute HMAC
- Data (encrypted), Data HMAC, Counter
  - Metadata Controller computes HMAC(Encrypted data, Counter)
  - Assert Data hash = HMAC(Encrypted data, Counter) computed to verify integrity of data
  - Decrypt data
  - Return

Design Assumptions

1. Channels between DRAM vaults have minimal latency
2. Channels between DRAM vaults are secure
3. Keys used for HMAC computation can be stored securely in the near data accelerators

Implementation

Preliminary Simulation: Baseline

- Assumes a configuration without metadata caching or NDP
- All metadata requests for a given data request are handled in the same vault
- All types of metadata are read and hashed (when applicable) sequentially

Preliminary Simulation: Optimization

- Meant to mimic metadata being processed with NDP
- All metadata are technically still being handled in the same vault as their corresponding data request
- Only HMAC + 1 tree node is added to the read queue
- Only 1 tree node is added to hash queue
    - Recursively marks all children as completed when its read and hash have finished
    - This mimics those child requests being processed in parallel

Experimentation

Simulator Configuration

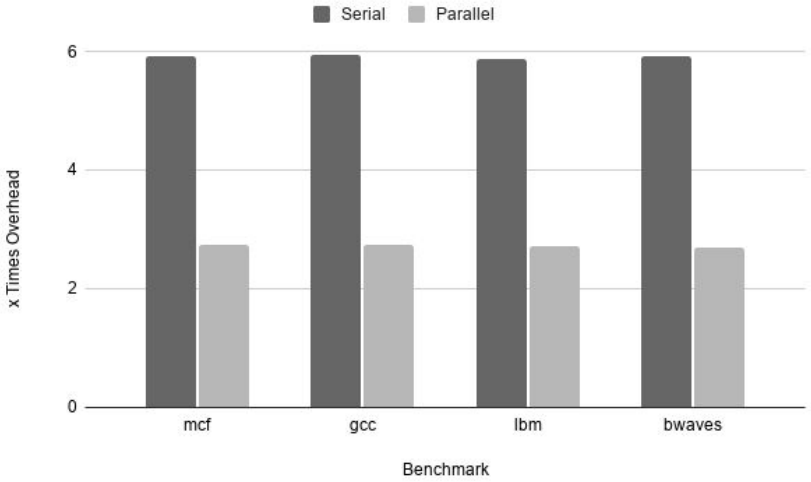
System Configuration	
CPU	1 in-order processor
L1 Cache	32kB instruction cache + 32kB data cache, both 2-way set associative
L2 Cache	512kB, 8-way set associative
Main Memory	16 DRAM vaults, 128MB each

Benchmarks

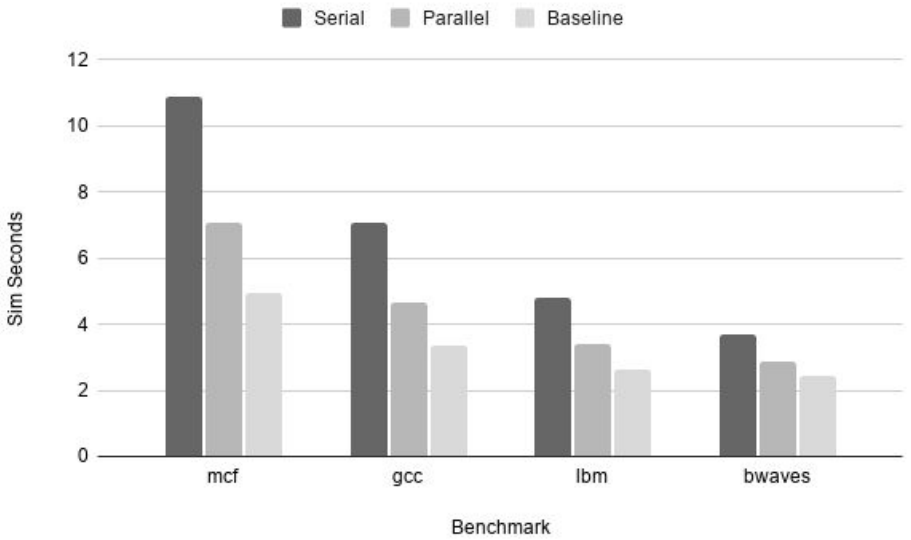
SPEC CPU 2017:

Benchmark	MPKI
mcf	21.0
gcc	12.9
lbm	7.8
bwaves	4.0

Approximate Results



The DRAM overhead of processing metadata serially rather than in parallel (using NDP) in comparison to a baseline implementation (non-secure memory). DRAM overhead is computer using the number of cycles a data request spends being processed in DRAM



Time required for Baseline (non-secure memory), serial (standard secure memory), and parallel (secure memory with NDP) implementations

Future Work

1) Reconfigure the simulator setup to actually model the proposed scheme and validate the initial approximations

2) Combine NDP with metadata caching

Summary

- Processing security metadata in memory has the potential to reduce overheads up to 50%
- Potential to further reduce overheads with Metadata Caching