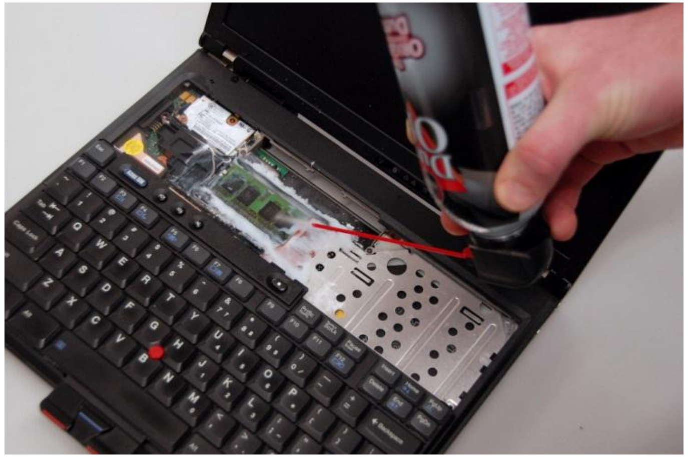
20th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2017)

Secure In-Cache Execution

Yue Chen, Mustakimur Khandaker, Zhi Wang
Florida State University

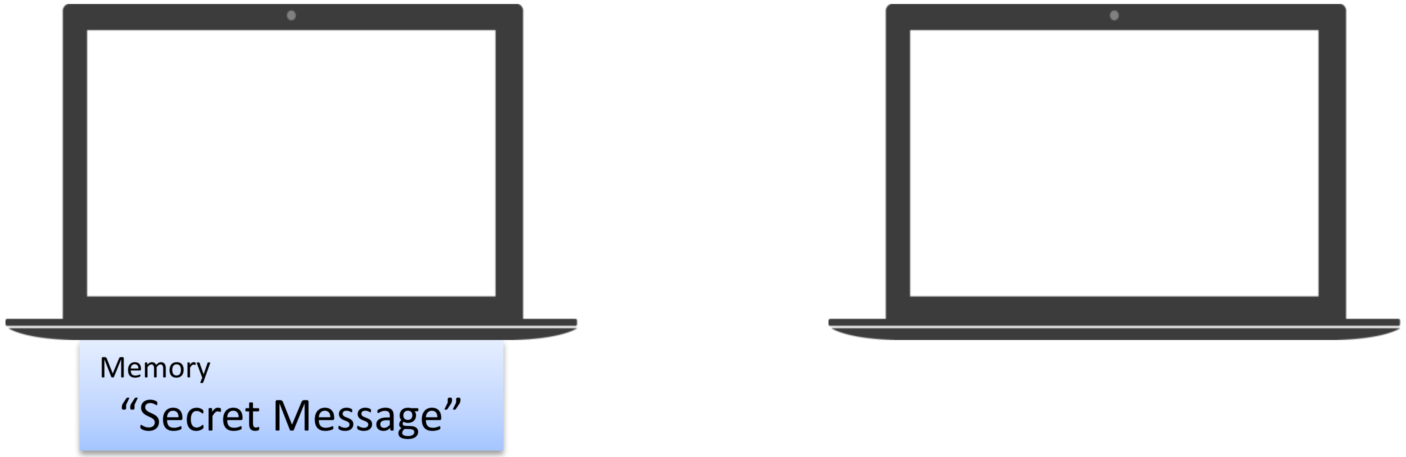
Cold Boot Attack

- Dump memory by freezing and transplanting
- Steal sensitive information



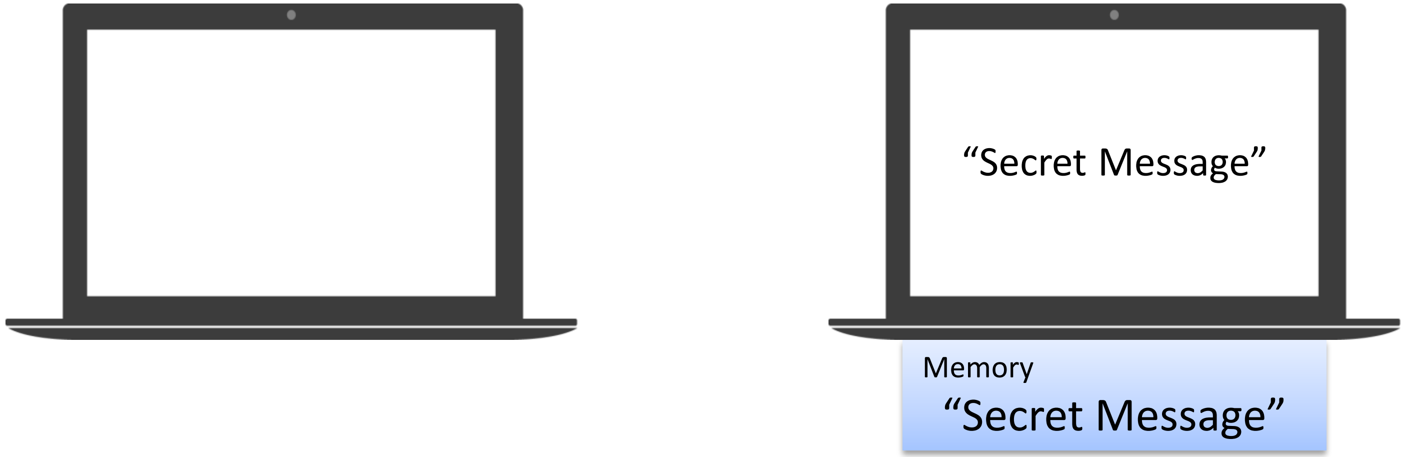
Cold Boot Attack

- Sensitive memory content in plaintext



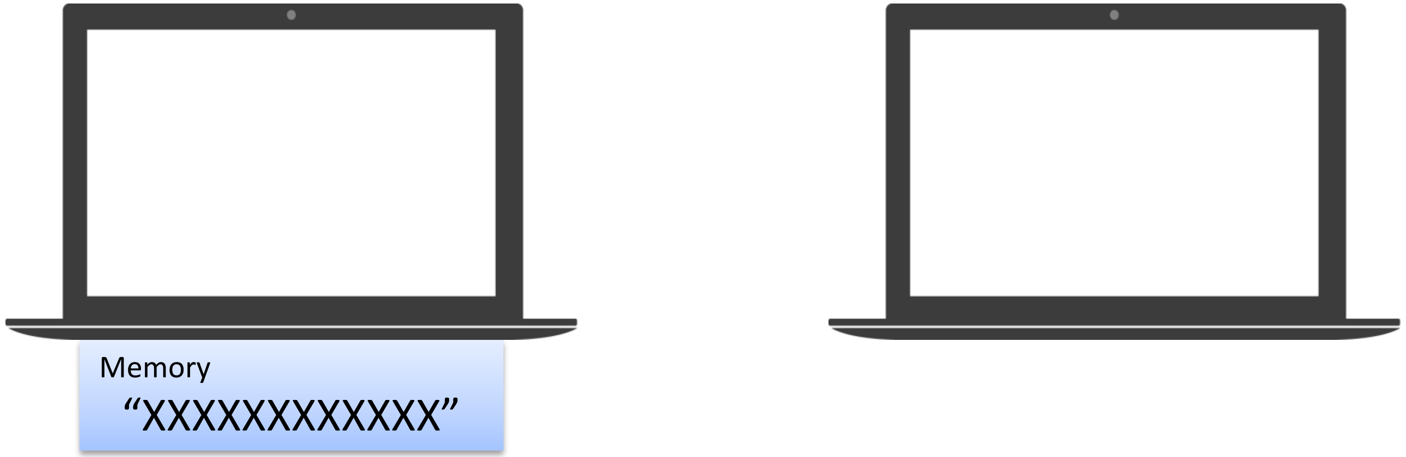
Cold Boot Attack

- Sensitive memory content in plaintext



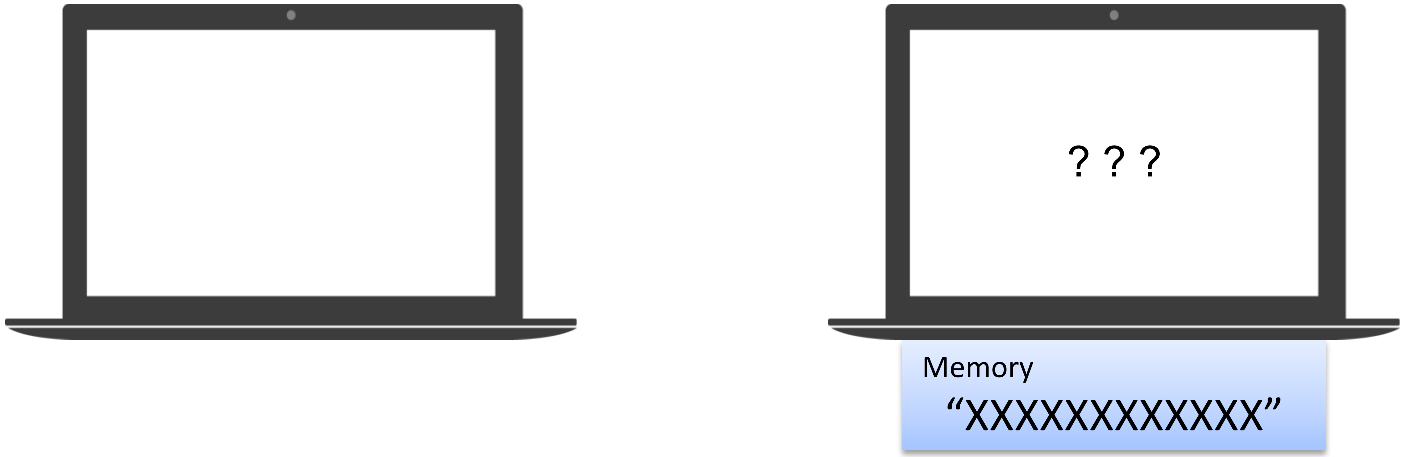
Our Solution

- Sensitive memory content cannot be read with encryption



Our Solution

- Sensitive memory content cannot be read with encryption



EncExec: Design Goals

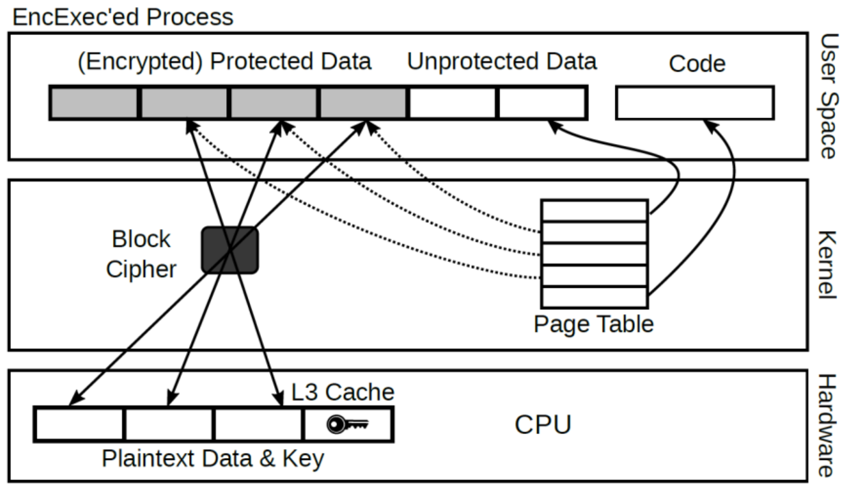
- Data secrecy
  - Plaintext view only in cache; key protected as well
- Performance isolation
  - Performance impact isolated from other processes
- Application transparency
  - User program unmodified to run under EncExec

Threat Model

- Able to perform cold boot attacks
- No malware installed (e.g., kernel rootkit)
- Typical use scenario:
  – Laptops lost in public places, even protected by encrypted hard disks and screen locks

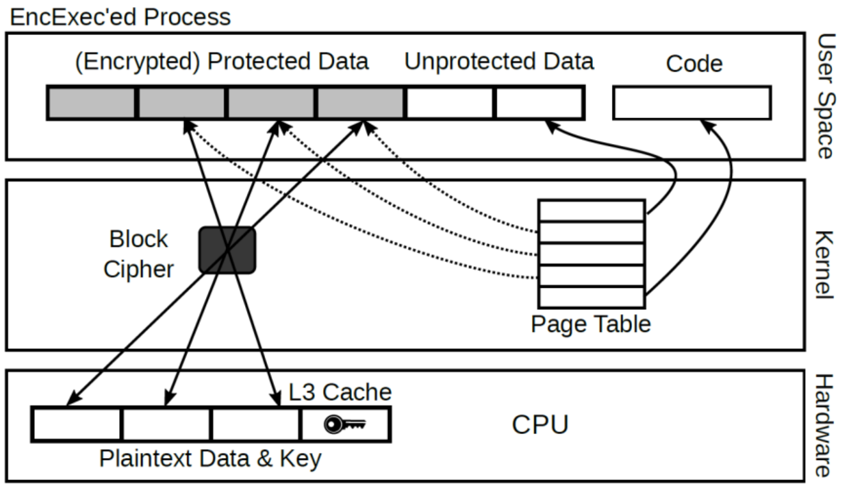
Design Overview

- Data in memory always encrypted; decrypted into the L3 cache only when accessed
- Use reserved cache as a window over protected data
  - Use L3 (instead of L1 or L2) cache to minimize performance impact



Design Overview

- Decrypted data will *never* be evicted to memory (no cache conflict)
  - Extend kernel's virtual memory management to strictly control access
  - Only data in the window are mapped in the address space
  - If more data than window size -> page replacement



Design Overview

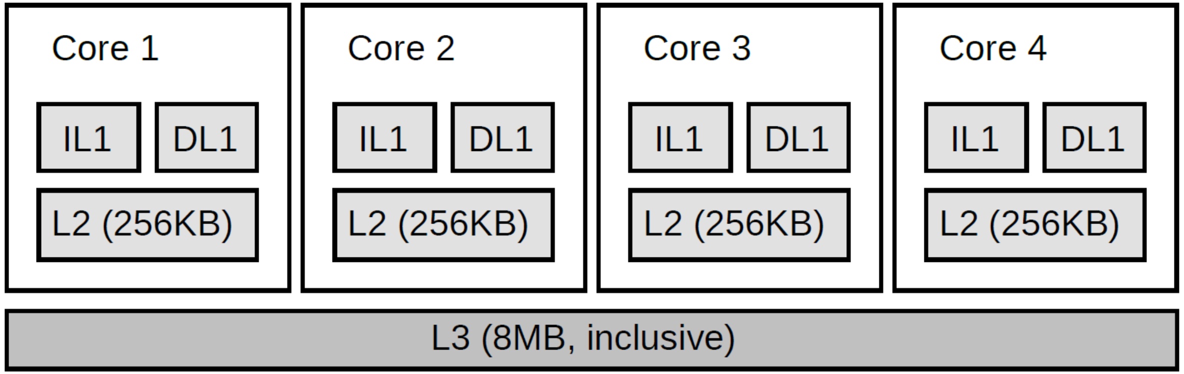
Two modes:

1. Given a block of secure memory for storing critical data
   - Need to (slightly) modify the program
2. Use reserved cache to protect all the data of the process
   - Use the reserved cache as a moving window

Design: Key Techniques

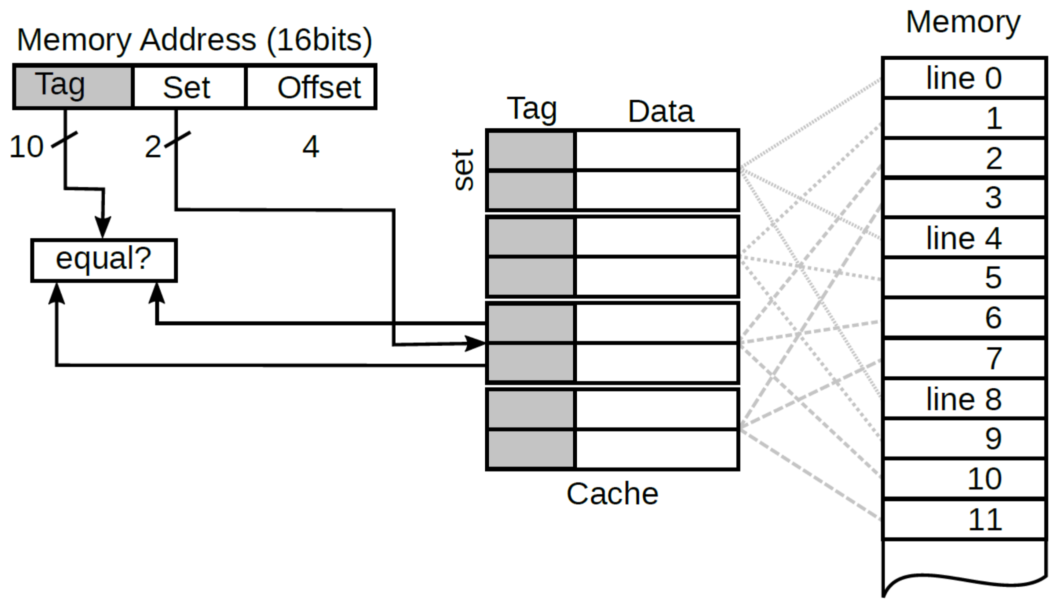
- Spatial cache reservation
  - Reserves a small part of the L3 cache for its use
- Secure in-cache execution
  - Data encrypted in memory, plaintext view only in cache

CPU Cache



Intel Core i7 4790 cache architecture

CPU Cache



2-way set-associative cache, 8 cache lines in 4 sets. Each cache line has 16 bytes.

Challenges: Spatial Cache Reservation

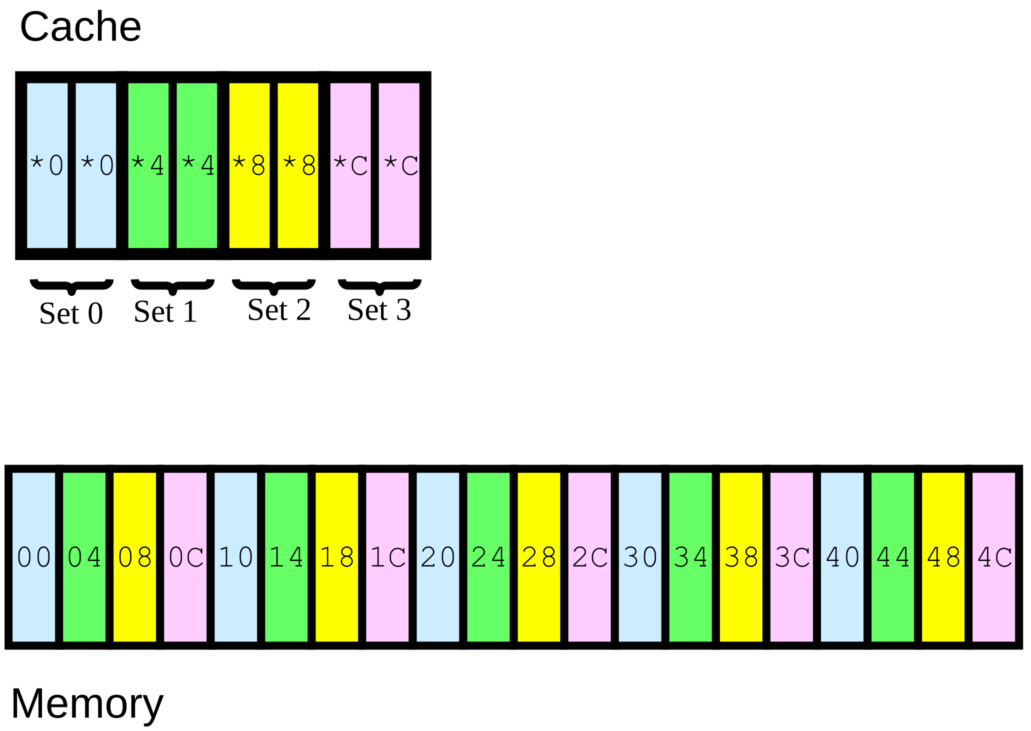
- Fine-grained cache control
  - x86 transparently manages cache assignment and replacement

- Countermeasures:
  - Rule 1
    - Protected data are only cached by the reserved cache
    - No other memory is cached by the reserved cache
  - Rule 2:
    - Accessible (decrypted) protected data is less than the reserved cache size
  - Thus, reserved cache content will not be evicted

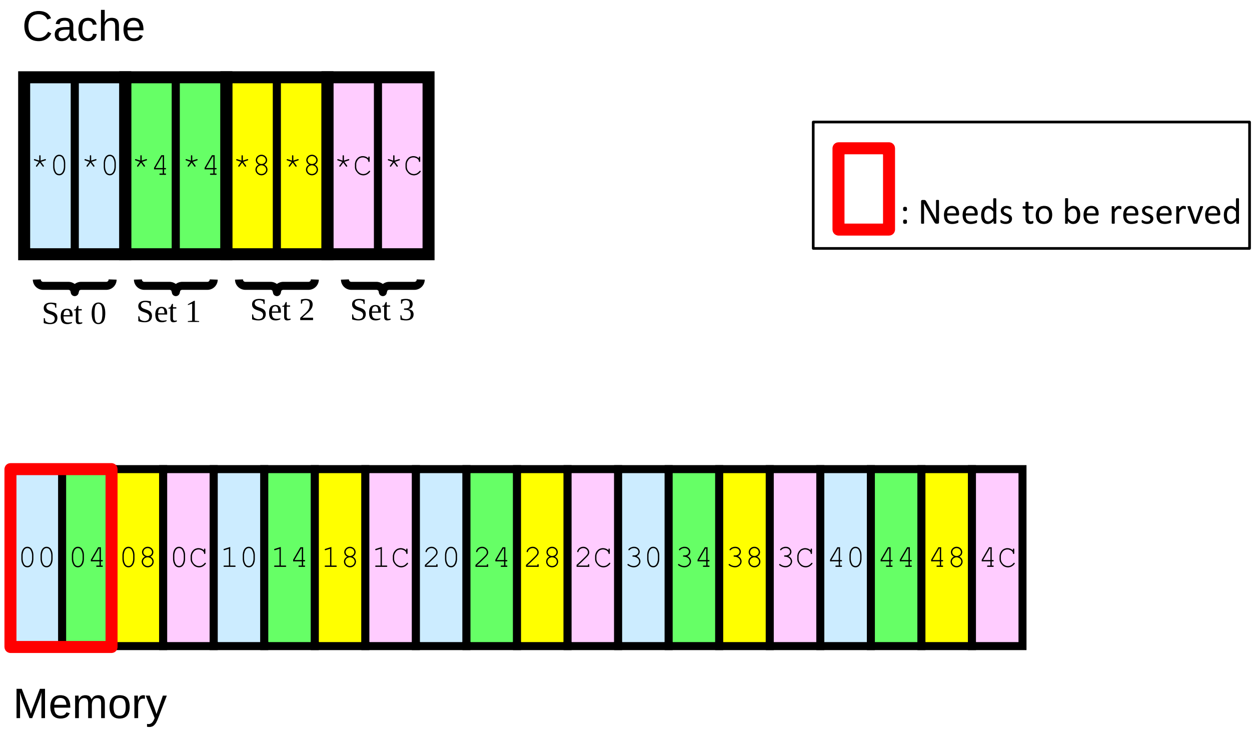
Design: Spatial Cache Reservation

- Use page table to control reserved memory usage
- Page table can only map page-sized and page-aligned memory
- Reserve at least a whole page on the L3 cache
- Reserve a smallest page of the cache (4KB)
  - How much space in total we need to reserve?

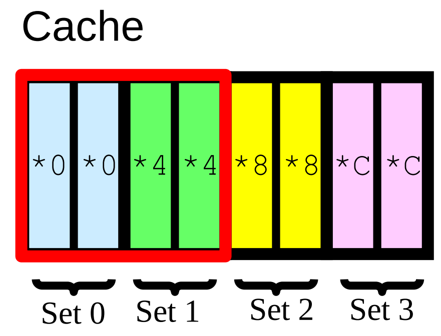
Design: Spatial Cache Reservation



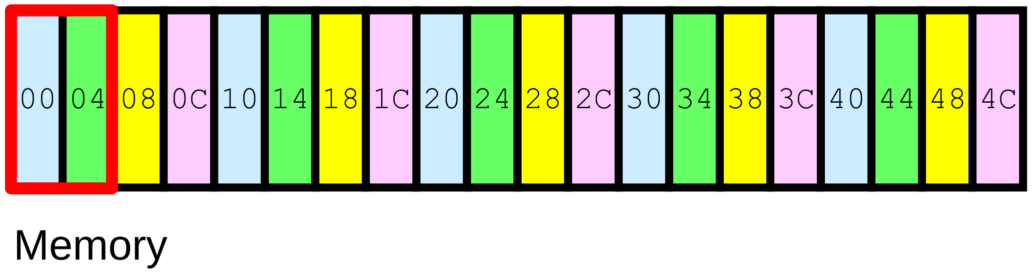
Design: Spatial Cache Reservation



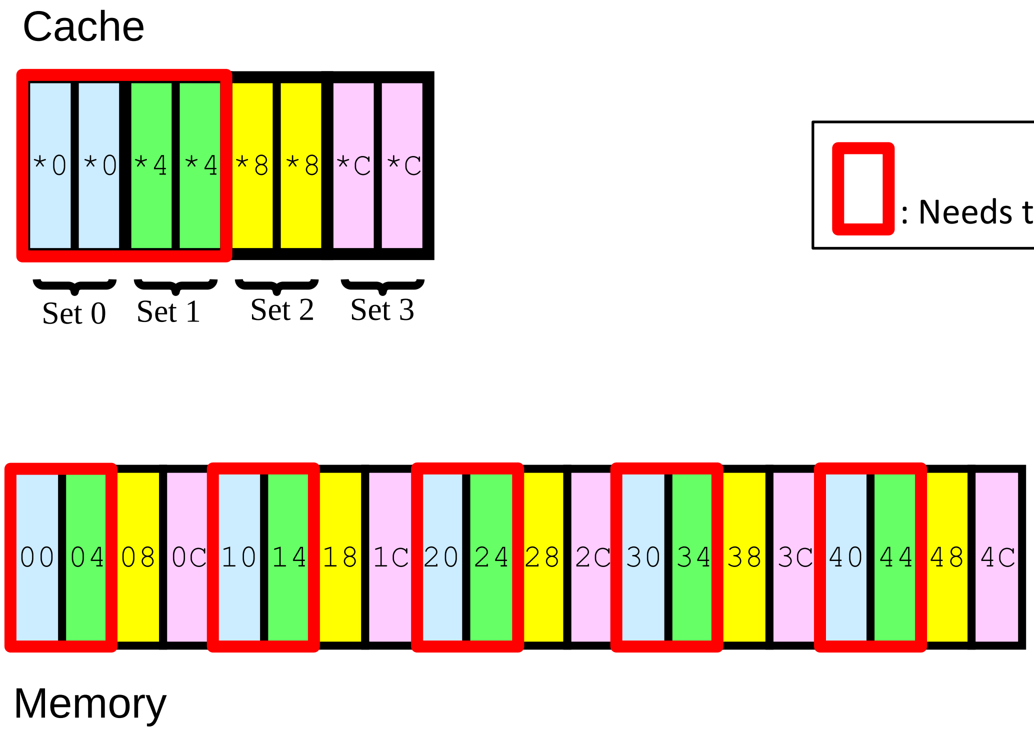
Design: Spatial Cache Reservation



: Needs to be reserved



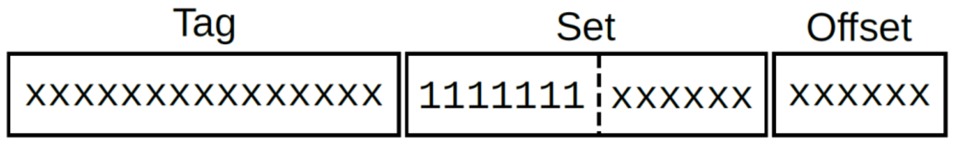
Design: Spatial Cache Reservation



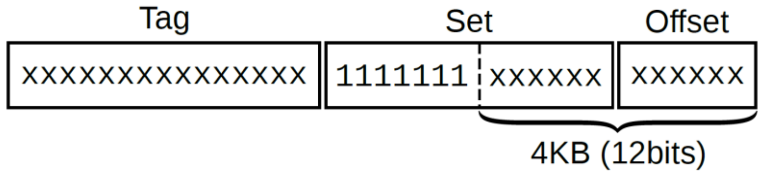
🟥 : Needs to be reserved

Example: Spatial Cache Reservation

- Intel Core-i7 4790 L3 cache
  - 16-way set-associative; physically indexed and physically tagged
  - Cache line size: 64 bytes = 2^6 bytes (`offset` field: 6 bits)
  - Cache size: 8 MB
- Set number: 8M/(64*16) = 8192 = 2^{13} (`set` field: 13 bits)
- If machine has 16GB (2^{34}) of physical memory, `tag` field has 15 bits (34 – 6 - 13 = 15).



Example: Spatial Cache Reservation



- Reserve one page (4KB)
- 64 cache lines in one page
  - Page_size/cache_line_size = 4K/64 = 64
- Need to reserve 64 cache sets
  - All the cache lines in the same set reserved together (16-way)
- Reserve 64KB cache in total
  - 64 (set number) * 16 (associativity ways) * 64B (cache line size) = 64KB

Example: Spatial Cache Reservation

- Reserve 1/128 of the physical memory
  - 64 (reserved sets) / 8192 (total) = 1/128
- Reserve one physical page for every 128 pages
- If RAM is 16GB, the total reserved memory is:
  - 16GB * 1/128 = 128MB
- Ensure no cache eviction:
  - Can use 64KB (16 pages) at a time of the 128MB
  - Name these 16 pages as plaintext pages
  - Protected data can be larger than 64KB as we use demand paging

Design: Secure In-Cache Execution

Desynchronize memory (encrypted) and cache (plaintext)

- Cache in write-back mode
  - Guaranteed by hardware and existing kernels (in most OS'es)
- L3 cache is inclusive of L1 and L2 caches
  - Guaranteed by hardware and existing kernels
- No conflict in the reserved cache
  - No more protected data at a time than the reserved cache size

Design: Secure In-Cache Execution

More data to protect?

- Demand paging
  - Access unmapped data -> page fault
  - Allocate a plaintext page (for securing data)
  - If no page available, select one for replacement
    - Encrypt the plaintext page, copy it back
    - Decrypt faulting page into plaintext, update page table if necessary

Design: Secure In-Cache Execution

- Dedicate one plaintext page to store keys and sub-keys
  - Cannot be evicted or replaced
- Frequent protected data encryption/decryption
  - Use CPU built-in support to speed up cryptographic algorithms

Implementation: Spatial Cache Reservation

- Reserve physical pages in the booting process
  - Modify allocators to skip reserved pages
- Make sure no reserved pages exist in page table
- Hook run-time page allocator and kernel’s whole-cache flushing function

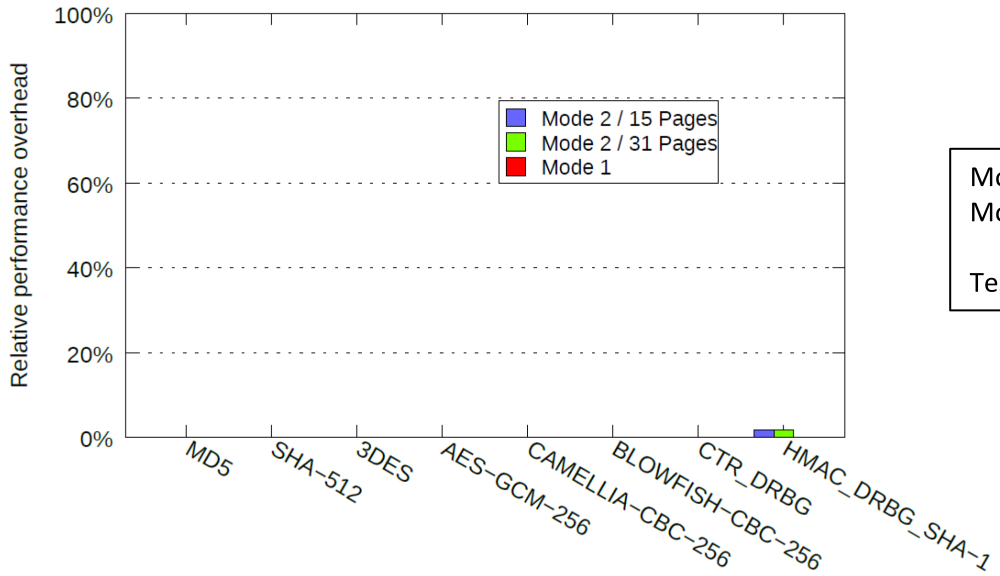
Implementation: Secure In-Cache Execution

- pmap is used to unmap a page
  - Consist of architecture-specific data and functions to manage the process' page table
  - Maintain a reverse mapping for physical pages
  - Track page usage information for page replacement
- Remove shared protected pages from other processes

Performance Evaluation

- Use the hardware-accelerated AES (AES-NI) to encrypt and decrypt data
- About 3µs on average to encrypt/decrypt 4KB data using 128-bit AES algorithm
  - 6µs to replace an existing page

Performance Evaluation

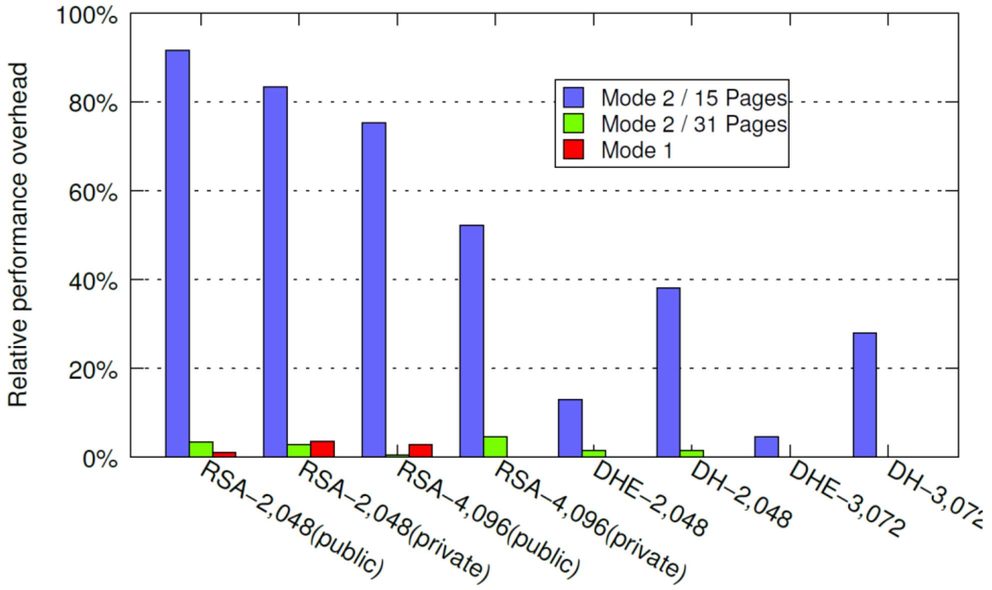


Overhead of common cryptographic algorithms

Mode 1: Choose data to encrypt
Mode 2: Encrypt all the data

Test with 15 or 31 plaintext pages

Performance Evaluation

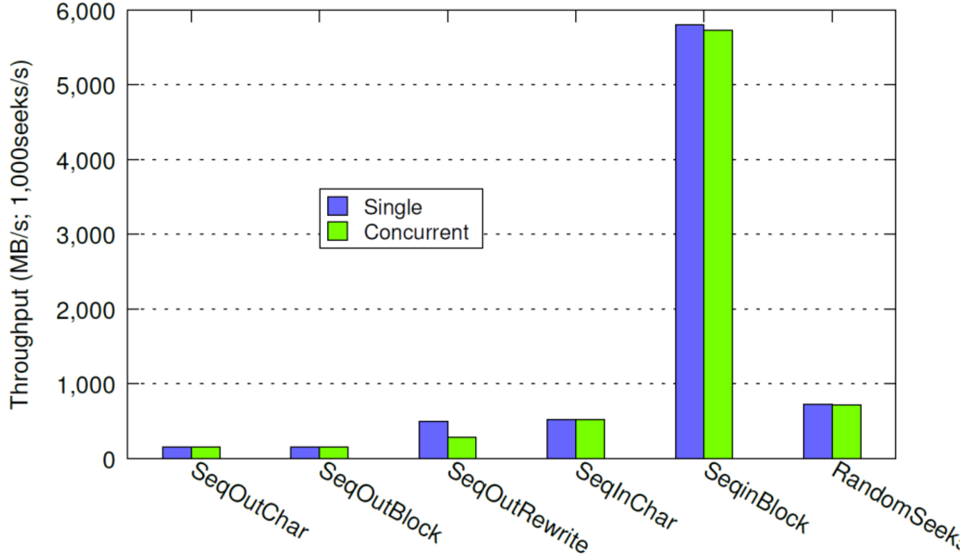


Overhead of RSA and DH handshakes

Mode 1: Choose data to encrypt
Mode 2: Encrypt all the data

Test with 15 or 31 plaintext pages

Performance Evaluation



Performance of Bonnie while concurrently running the mbed TLS benchmark. The unit on the Y-axis is MB/sec or thousand_seeks/sec (for RandomSeeks only).

Q & A

Secure In-Cache Execution

www.YueChen.me

Backup Slides

Compared to Intel SGX

- SGX is great!
- EncExec
  - Works on old CPUs
  - No time-consuming context switch
  - Supports unmodified programs