

2022-2023学年春季学期

计算机体系结构安全
Computer Architecture Security

授课团队：侯锐、朱子元、宋威
助 教：杨正帮

计算机体系结构安全

Computer Architecture Security

[第8次课] 缓存侧信道攻击及防御

授课教师：宋威

授课时间：4月11日

内容概要

- 缓存结构简述
 - 缓存结构、虚实地址转换、缓存一致性
- 针对L1的缓存侧信道
 - 基本攻击
 - Prime+Probe、Flush+Reload
- 针对LLC的缓存侧信道
 - Complex addressing、构造eviction set
- 防御措施
 - 缓存隔离
 - 缓存随机化

内容概要

○缓存结构简述

- 缓存结构、虚实地址转换、缓存一致性

○针对L1的缓存侧信道

- 基本攻击
- Prime+Probe、Flush+Reload

○针对LLC的缓存侧信道

- Complex addressing、构造eviction set

○防御措施

- 缓存隔离
- 缓存随机化

○缓存

○片上的高速存储

○缓解高速处理器流水线和低速外部存储器之间的速度差

○存储具有局部性的数据

○空间局部性、时间局部性、流局部性

○组织形式

○Directly mapped (直连)

○Fully associated (全相连)

○Set way associated (多路组相连)

○层次

○L1-I, L1-D: 私有高速

○(L2): 统一, 单核共享

○LLC: 统一, 多核共享

○包含关系

○Inclusive, non-inclusive, exclusive (victim cache)

○虚拟地址空间

- 为所有进程提供统一的虚拟地址空间

- 合理利用有限物理内存的中间层

- 分配方式

 - 按页分配，单页4KB

 - 页表存储，多级翻译机制

 - PTW: (hardware) page table walker

 - TLB: Translation lookaside buffer

- 缓存种类

 - PIPT: **Physically indexed physically tagged (LLC)**

 - VIPT: **Virtually indexed physically tagged (L1/L2)**

 - VIVT: Virtually indexed virtually tagged (L1)

- 别名(alias)

 - 同一个物理地址被多个虚拟地址映射（同一虚拟地址映射多物理地址）

 - 页着色 (page coloring)

○Cache coherency

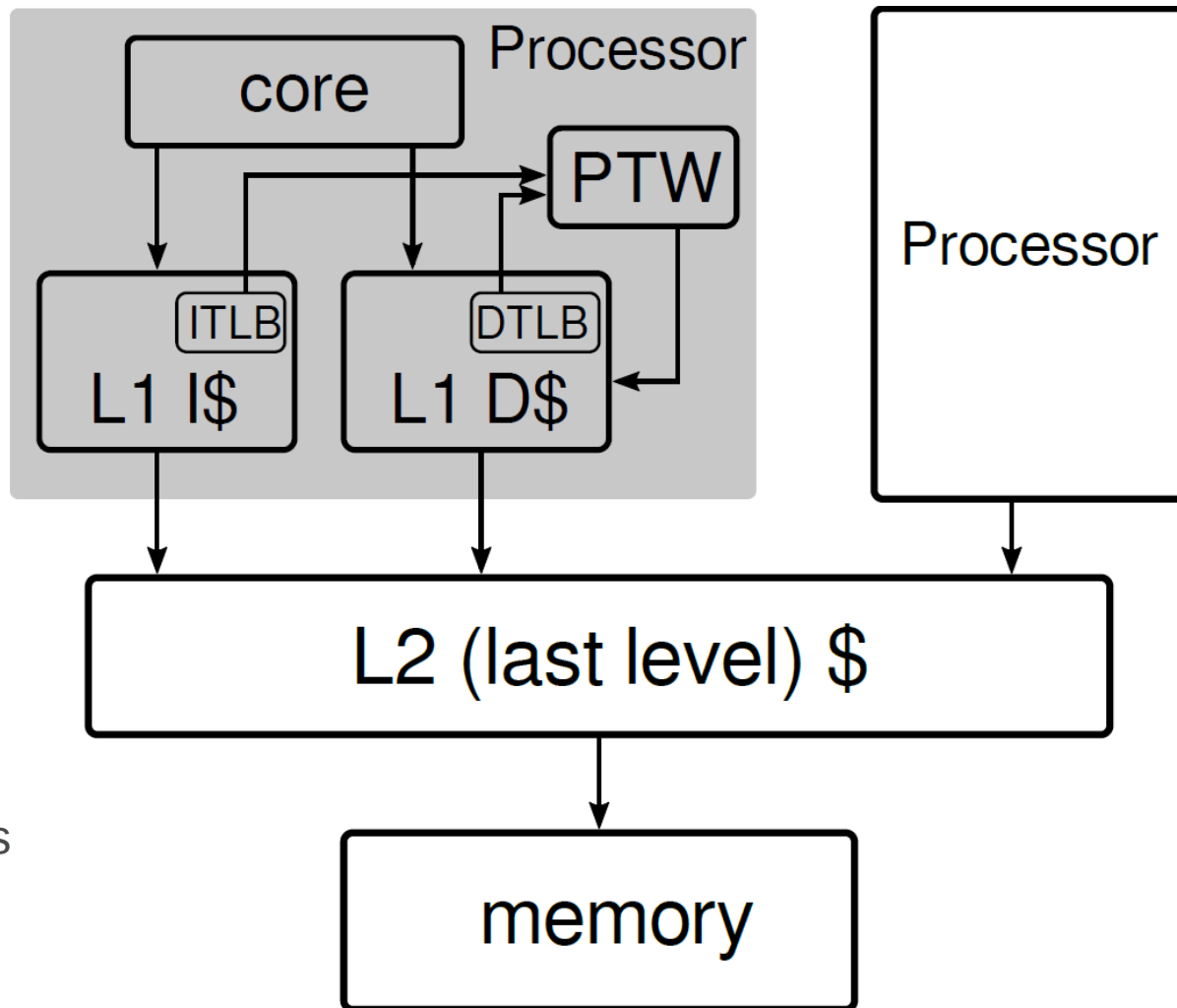
- 在多层/多个缓存之间保持同一数据的及时更新。
- MESI: modified, exclusive, shared, invalid
- Inclusive, non-inclusive, exclusive

○Cache consistency

- 在多层/多个缓存之间保持不同数据更新的合理顺序
- Program order, STO (Total store order), weak order
- 原子操作
 - 原子指令 (单一数据的读写)
 - 单副本 (LLC write-through)
 - barrier (STO, 影响多核coherence)
 - 回退 (LR/SC)

对于基本的缓存侧信道, 理解inclusive cache coherency就基本够了。

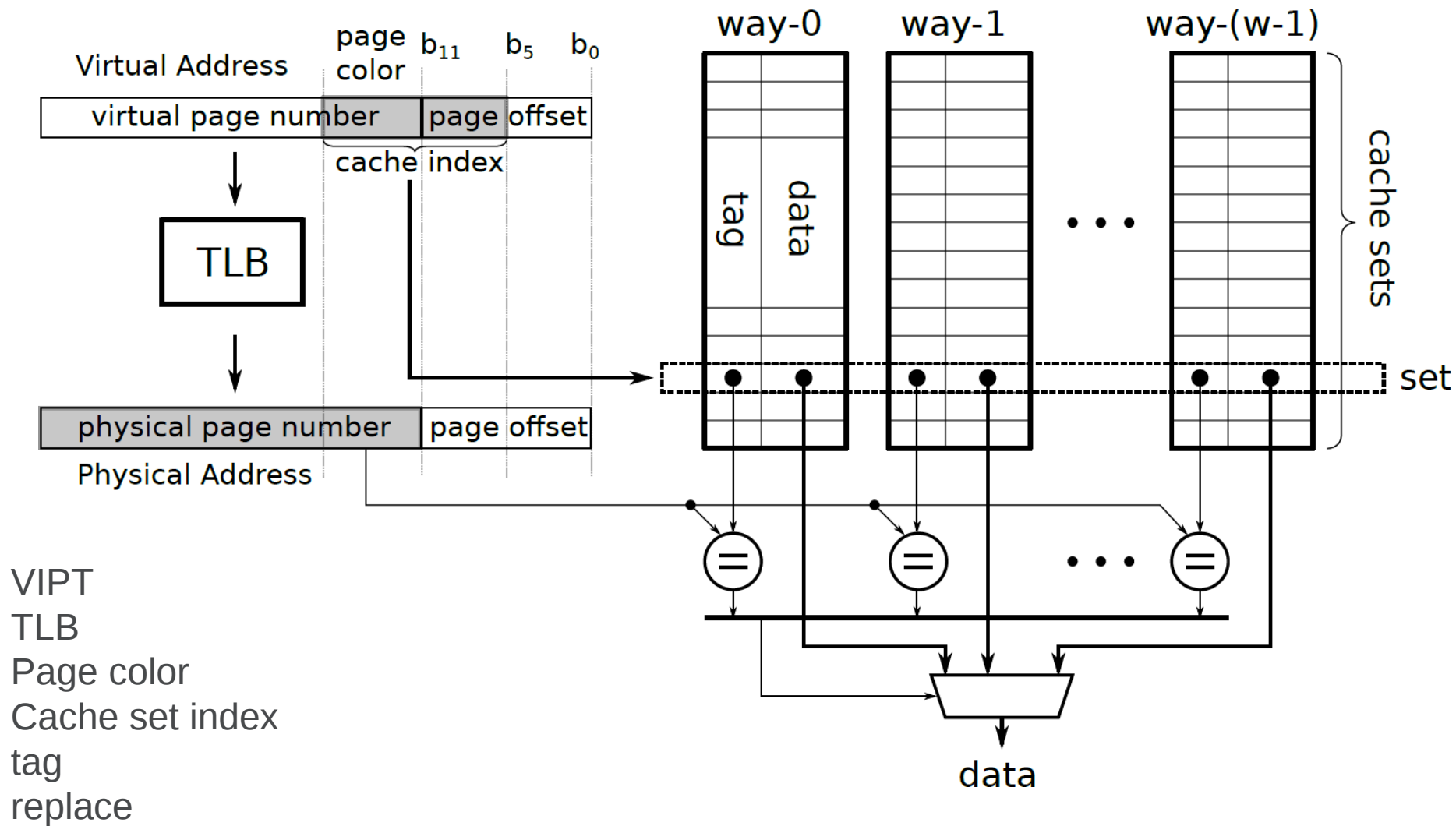
缓存的基本结构



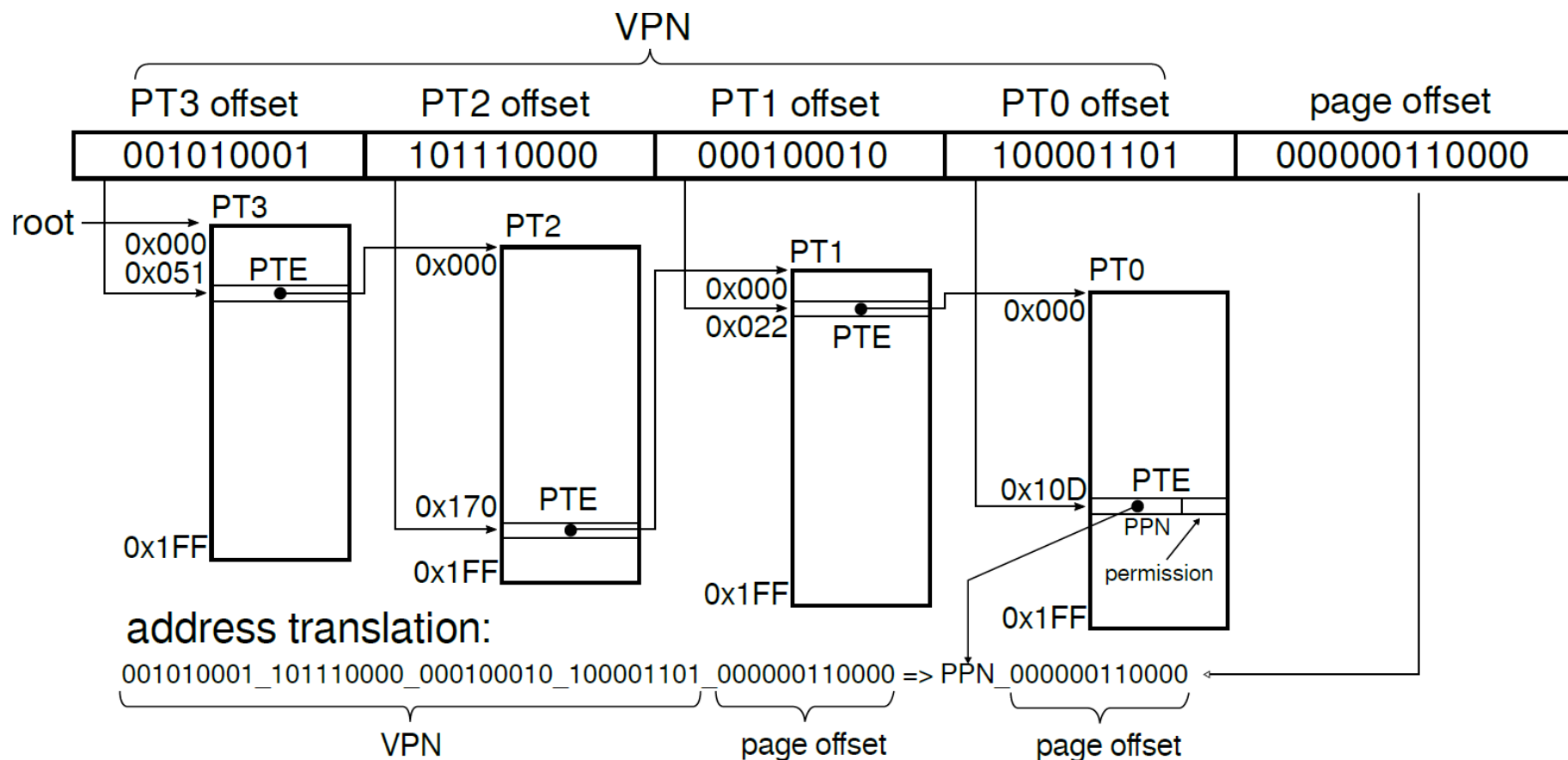
\$: cache

层次结构
Inclusiveness
TLB/PTW
Coherence

缓存的基本结构



缓存的基本结构



PTE, VPN, PPN
虚拟化

内容概要

- 缓存结构简述
 - 缓存结构、虚实地址转换、缓存一致性
- 针对L1的缓存侧信道
 - 基本攻击
 - Prime+Probe、Flush+Reload
- 针对LLC的缓存侧信道
 - Complex addressing、构造eviction set
- 防御措施
 - 缓存隔离
 - 缓存随机化

缓存侧信道攻击的前提条件

○共享

- 攻击者和被攻击者共享一个缓存

○可控

- 攻击者可以控制缓存状态
 - 粗：是否缓存
 - 细：一个缓存块是否被缓存

○可测

- 缓存块是否被缓存 (miss or hit)
- Miss in ? L1, L2, LLC, TLB, L2-TLB

○可推断

- 从缓存状态可反推信息
 - 一个页是否存在
 - 一个缓存块是否被访问

缓存侧信道攻击的前提条件

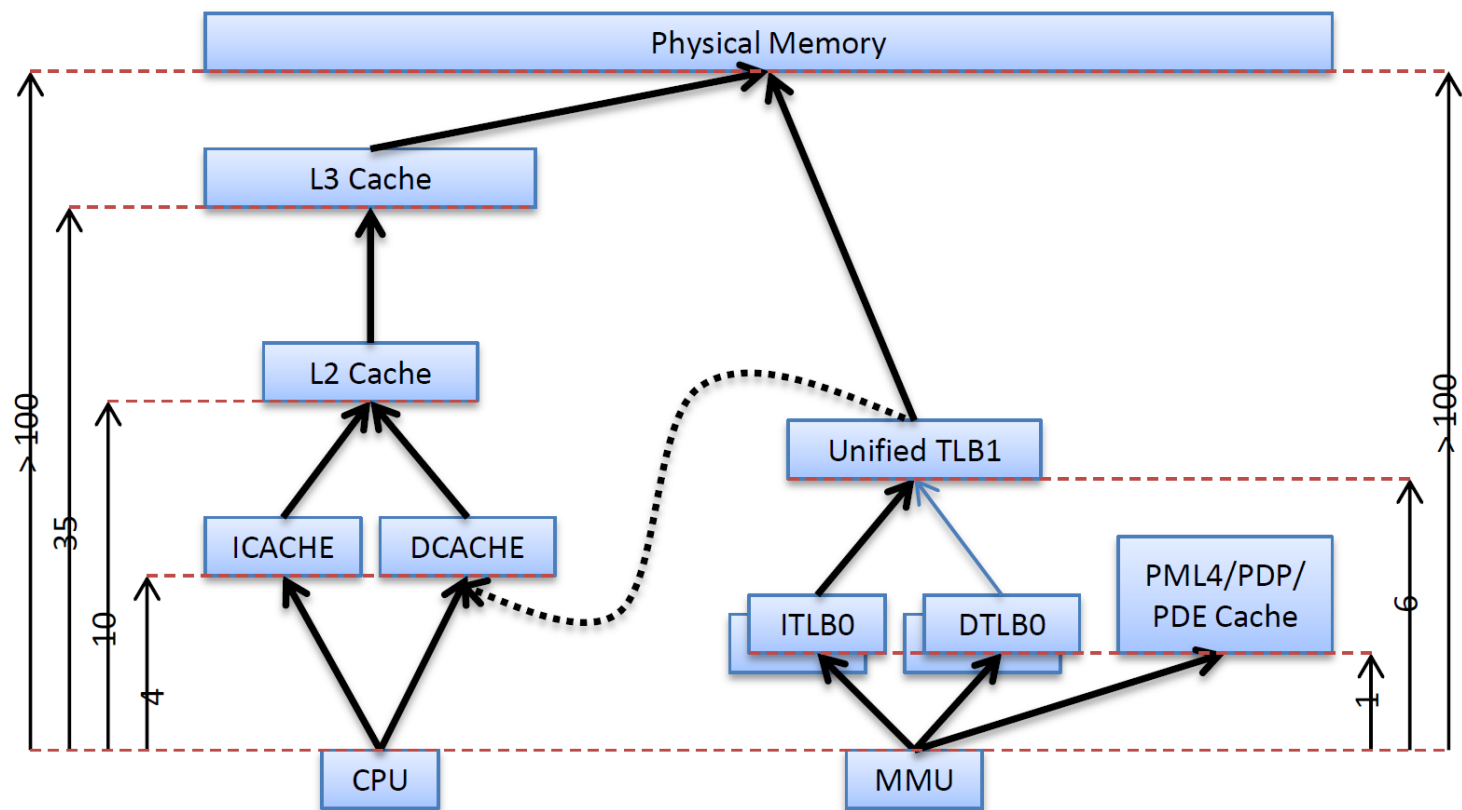


Figure 2. Intel i7 memory hierarchy plus clock latency for the relevant stages (based on [32], [33])

Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space ASLR. IEEE Symposium on Security and Privacy, 2013, pp. 191—205.

○两步攻击

○第一步：布置攻击

将缓存目标调整为一个受控状态

- 将被攻击者的数据从一个缓存set清除

○第二步

观察缓存变化

- 让被攻击者执行
- 观察缓存目标的状态是否有变化
被攻击者将一个新数据放入目标set?

**如果有变化，缓存目标和被攻击程序的执行相关
反推关键信息**

○目标

○被攻击程序是否读取了位于0x80086400处的数据？

○第一步：布置攻击

将0x80086400从缓存中清除

○第二步

让被攻击程序执行

测量读取0x80086400数据的时间

如果时间短，被攻击程序读取了该数据（缓存命中）

如果时间长，被攻击程序没有读取（缓存不命中）

第一步和第二步的具体方式决定了侧信道攻击的方法。

○ Flush+Reload

○ 被攻击程序是否读取了位于0x80086400处的数据？

○ 第一步：Flush

使用cflush指令，将0x80086400从缓存中清除

○ 第二步：Reload

让被攻击程序执行

测量读取0x80086400数据的时间

如果时间短，被攻击程序读取了该数据（缓存命中）

如果时间长，被攻击程序没有读取（缓存不命中）

攻击者是怎么刷走了被攻击者的数据？？

○ Prime+Probe

○ 被攻击程序是否读取了位于0x80086400处的数据?

○ 第一步: Prime

使用驱逐集 (eviction set) , 将0x80086400从缓存中挤出
0x?????400

{0x80085400, 0x80086400, 0x80087400, 0x80088400,
0x80095400, 0x80096400, 0x80097400, 0x80098400}

○ 第二步: Probe

让被攻击程序执行
测量读取驱逐集的时间

如果时间短, 被攻击程序没有访问该数据 (eviction set缓存命中)

如果时间长, 被攻击程序读取了该数据 (eviction set缓存部分不命中)

攻击比较

	Flush+Reload	Prime+Probe
第一步	Flush攻击地址	装载驱逐集到目标set
第二步	读取攻击地址	读取驱逐集
泄露的信息量	缓存块地址	缓存set index
地址类型	虚拟地址	虚拟地址/物理地址
特权	Flush权力	无
数据共享	被攻击者地址可读取	无
进程限制	共享被攻击数据	共享缓存
速度	快	慢

○具体数据

- 加解密算法
- 密钥信息
- 其他关键信息
- 秘密通讯 (covert channel)

○页表项

- 用户态ASLR, 猜VA
- 系统ASLR, 破解系统空间
- SGX, 数据使用

内容概要

- 缓存结构简述
 - 缓存结构、虚实地址转换、缓存一致性
- 针对L1的缓存侧信道
 - 基本攻击
 - Prime+Probe、Flush+Reload
- 针对LLC的缓存侧信道
 - Complex addressing、构造eviction set
- 防御措施
 - 缓存隔离
 - 缓存随机化

○跨核攻击

- 被攻击者是运行在另外一个核上的进程。
- 攻击者是运行在另外一个核上的VM。

○逃避现有防御

- 关闭SMT
- 上下文切换时清除L1
- 缓存隔离（不同进程不共享缓存块）
- Flush指令被禁止

LLC是多核共享的缓存，其分配由硬件控制，远离软件，反而不容易避免共享。

○物理地址

○LLC使用物理地址索引，不能使用虚拟地址直接攻击

- 破解page mapping（早期系统直接暴露给用户）

- 通过扫描撞地址

○复杂地址寻址模型(complex address scheme)

- LLC是分片的

- 物理地址到LLC片的映射由一个不公开的哈希函数决定

- 基于Prime的攻击可能需要破解该哈希函数

Eviction Set

○Eviction Set

- 包含了多个缓存块
- 其中包含N个缓存块都映射到同一个目标缓存组
- 按照某一个访问顺序访问Eviction Set可以将目标缓存组中的任意原有缓存块替换出目标缓存
- N是映射到目标缓存组的缓存块的最小个数

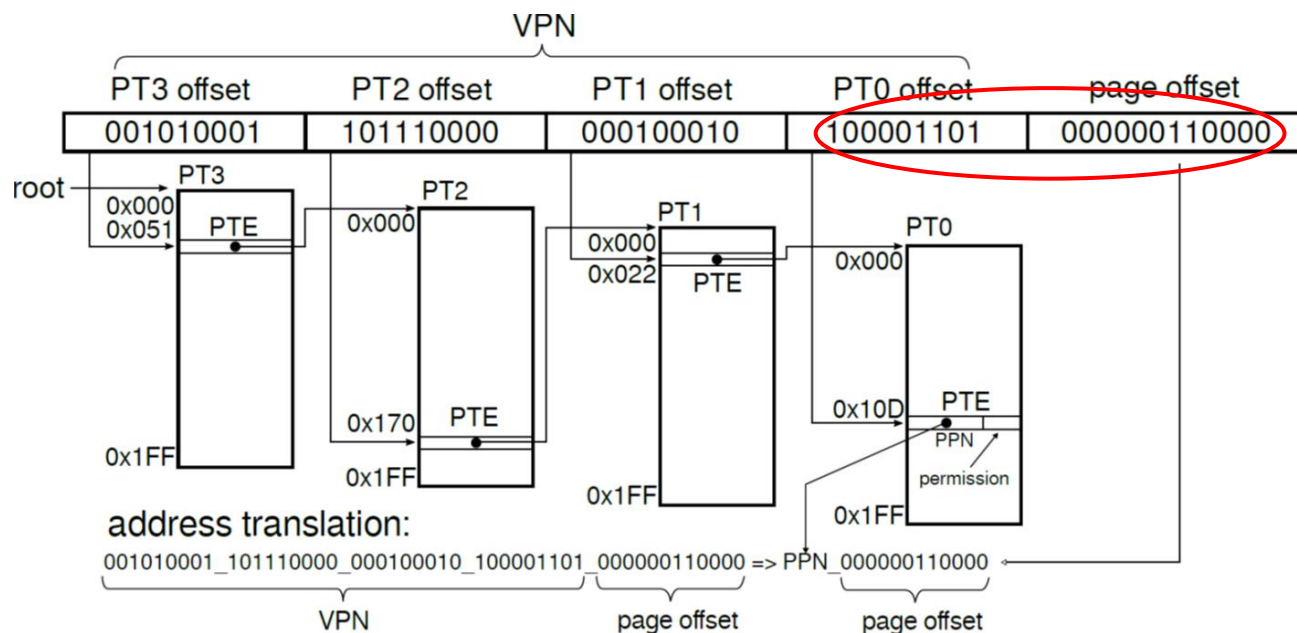
○最小Eviction Set

- Eviction Set的缓存块个数为N
- 一般情况下, $N=W$, 缓存的路数

构造LLC的Eviction Set (1)

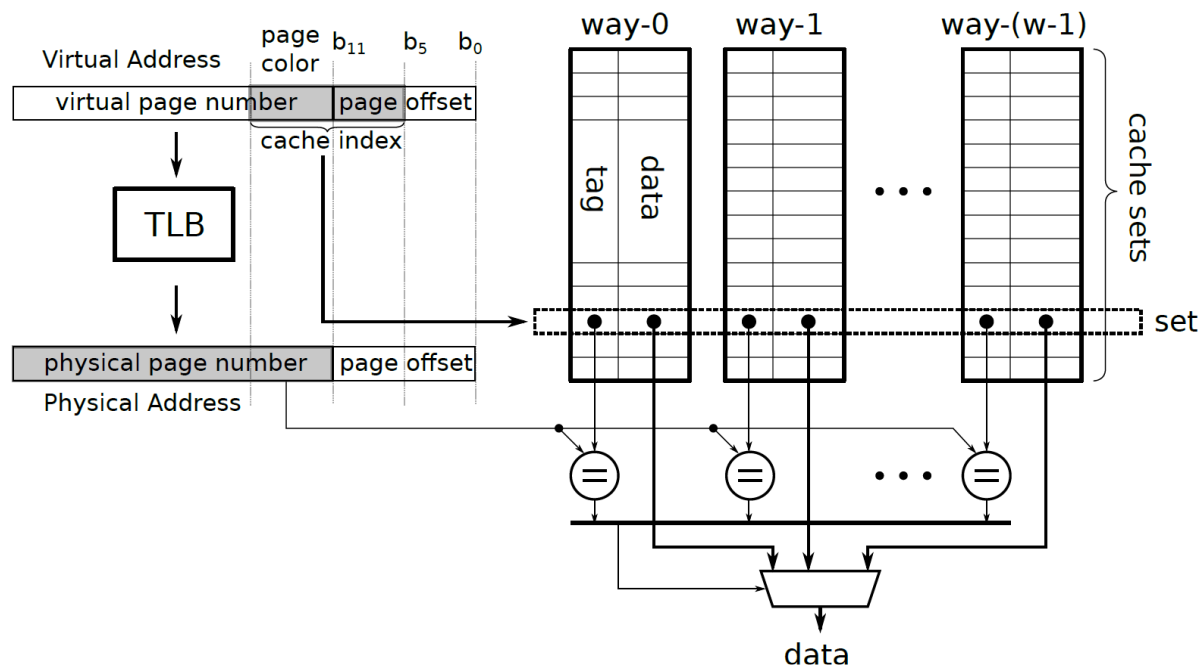
○大页 (Large/Huge page)

- 为了减少TLB的频繁缺失，应用程序可以申请大页
- 一个大页一般是2M（少一级页表）
- 攻击者控制了低21比特（虚拟物理相同）



构造LLC的Eviction Set (2)

- 大页让攻击者实际上获得了物理地址
 - LLC的大小在大页的级别 (2~16M)
 - Cache index大部分 (甚至全部) 落在了可控的21比特上
 - 没有complex address scheme的话, 已经可以随意构造Eviction Set



○绕过Complex Address Scheme

- 假设LLC被分成4个slice, 8路组相连 ($N=W=8$)
- 攻击者获得了>32个大页, 可构造关于任意地址 x 的一个eviction set C
- 将 C 化简成一个最小Eviction Set

Input: C , candidate set; x , target address.

Output: Minimal eviction set for x .

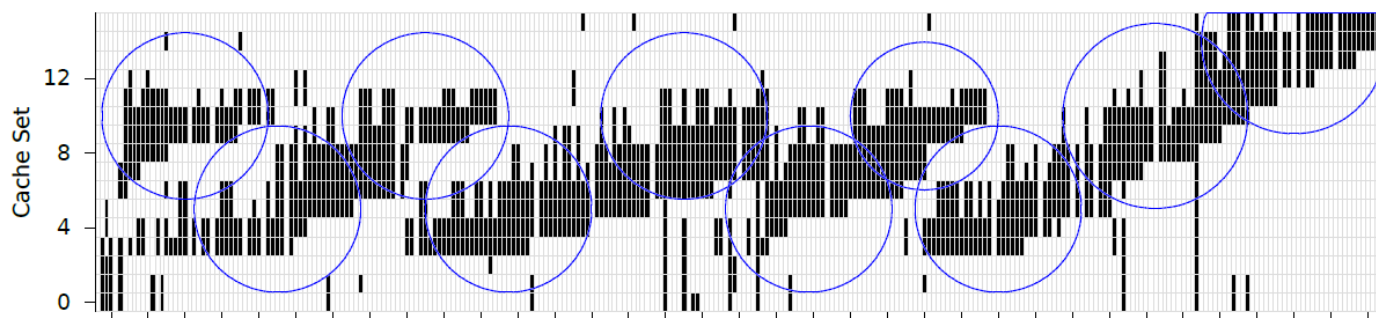
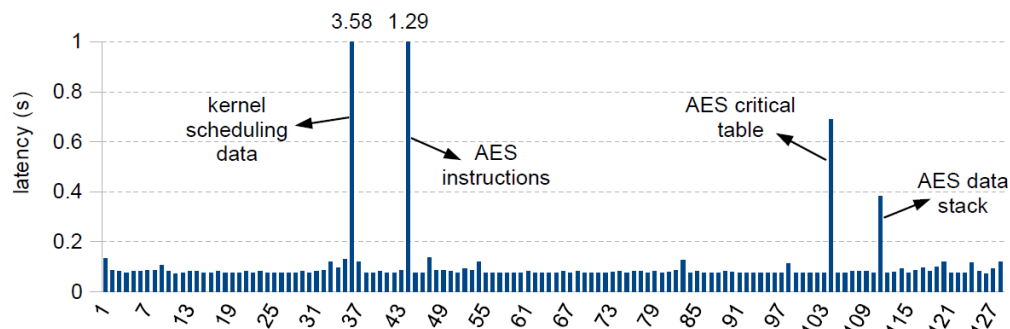
```
function prune_base( $C, x$ )  
  foreach  $e$  in  $C$  do  
    if  $\text{test}(C \setminus \{e\}, x)$  then  
       $C \leftarrow C \setminus \{e\}$   
    end  
  end  
  return  $C$   
end
```

- 有了众多的最小Eviction Set
- 假设攻击者已经破解page mapping
- 通过最小Eviction Set可以获得映射到一个缓存组的多个地址
- 由于page mapping已知，这些地址是物理地址
- 哈希方程
 - 一个64bit到2比特的映射（4个slice）
 - $\text{slice} = \text{hash}(\text{address})$
- 找到64个映射到同一缓存组的地址就可以破解一个哈希方程

		Address Bit																															
		3 7	3 6	3 5	3 4	3 3	3 2	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	0 7	0 6
2 cores	o_0						⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
4 cores	o_0					⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1				⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕		⊕			⊕	
8 cores	o_0		⊕	⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1	⊕		⊕	⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕		⊕			⊕	
	o_2	⊕	⊕	⊕	⊕			⊕	⊕			⊕	⊕			⊕	⊕			⊕			⊕			⊕	⊕				⊕		

如何确定被攻击者的地址

- 扫描缓存组，确定相关内存页，缓存组
- 进一步分析相关组，确定时间信息



Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In Proceedings of the 53rd Annual Design Automation, 2016.

内容概要

- 缓存结构简述
 - 缓存结构、虚实地址转换、缓存一致性
- 针对L1的缓存侧信道
 - 基本攻击
 - Prime+Probe、Flush+Reload
- 针对LLC的缓存侧信道
 - Complex addressing、构造eviction set
- 防御措施
 - 缓存隔离
 - 缓存随机化

缓存侧信道攻击的前提条件

○共享

- 攻击者和被攻击者共享一个缓存

○可控

- 攻击者可以控制缓存状态
 - 粗：是否缓存
 - 细：一个缓存块是否被缓存

○可测

- 缓存块是否被缓存 (miss or hit)
- Miss in ? L1, L2, LLC, TLB, L2-TLB

○可推断

- 从缓存状态可反推信息
 - 一个页是否存在
 - 一个缓存块是否被访问

○缓存隔离

- 缓存侧信道的前提条件之一是共享缓存
- Flush+Reload方式还需要共享内存
- 如果能不共享缓存/内存，则不能组织侧信道攻击

○Flush+Reload

- 关闭SMT
- 上下文切换时清除L1
- 关闭deduplication

- 关闭大页支持?
- 关闭flush支持?

缓存隔离 (2)

○ Prime+Probe (LLC)

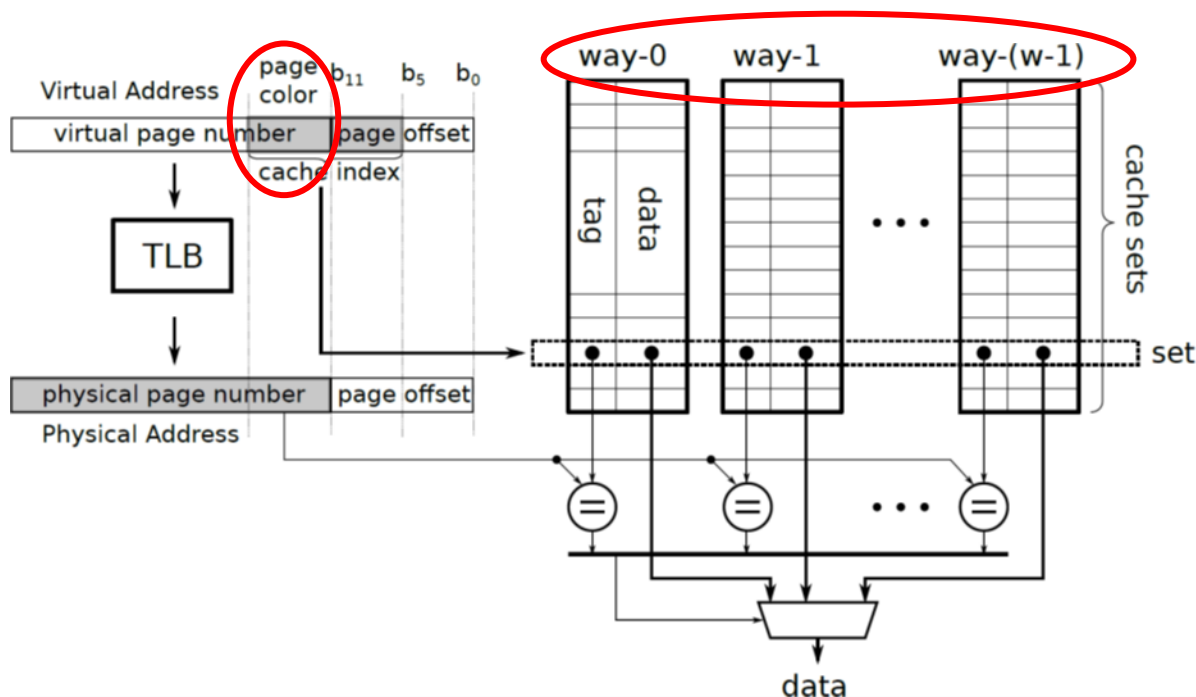
○ 缓存按路分割

- 给每个core分配不同的缓存路(way)

- Intel CAT

○ 缓存按着色分割

- 为每个进程/VM分配不同的page color



○缓存隔离

- 如果攻击者和被攻击者不共享缓存/内存，那么侧信道攻击就不能发生。
- 缓存隔离造成了资源的不合理分配
- 当攻击者和被攻击者不能被区分时，缓存隔离无效
 - 攻击者和被攻击者同属一个进程
 - 攻击者和被攻击者共享一个第三方库

缓存侧信道攻击的前提条件

○共享

- 攻击者和被攻击者共享一个缓存

○可控

- 攻击者可以控制缓存状态
 - 粗：是否缓存
 - 细：一个缓存块是否被缓存

○可测

- 缓存块是否被缓存 (miss or hit)
- Miss in ? L1, L2, LLC, TLB, L2-TLB

○可推断

- 从缓存状态可反推信息
 - 一个页是否存在
 - 一个缓存块是否被访问

○降低时钟精度

- 如果攻击者不能准确测量内存访问的时间差，则不能反推数据的缓存状态。
- 该防御已经在浏览器沙盒中大量部署（只不过目的是为了防御 Spectre）
- 操作系统和很多用户态软件大量依赖于高精度的时钟，并不能简单的通过降低时钟精度（甚至去除时钟来源）来组织一般的缓存侧信道攻击。
- 此外，攻击者也可能通过多线程在沙盒中自行构造高精度时钟。
 - Javascript不支持直接多线程编程
 - 但是攻击者也许可以通过worker的方式变相实现多线程

缓存侧信道攻击的前提条件

○共享

- 攻击者和被攻击者共享一个缓存

○可控

- 攻击者可以控制缓存状态
 - 粗：是否缓存
 - 细：一个缓存块是否被缓存

○可测

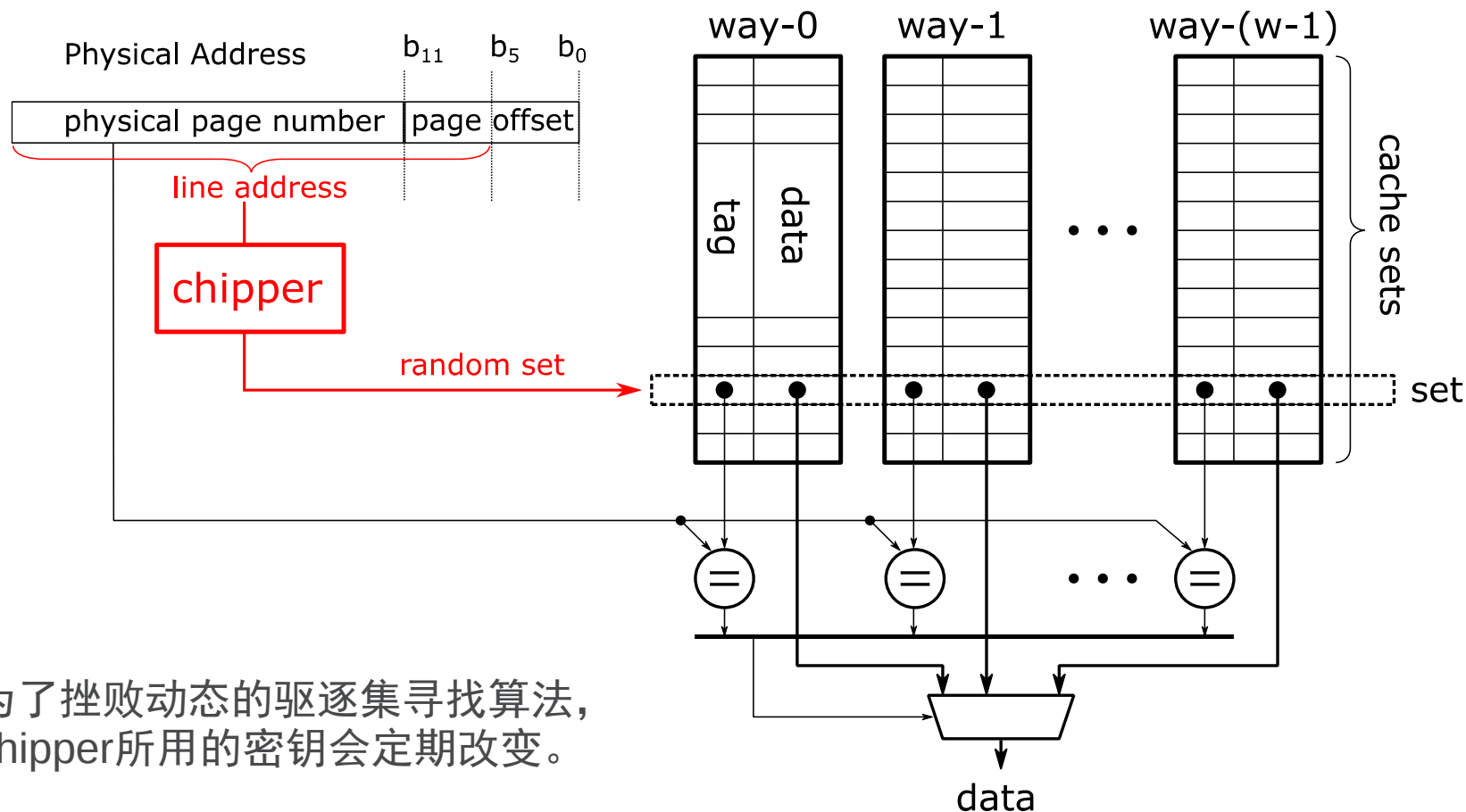
- 缓存块是否被缓存 (miss or hit)
- Miss in ? L1, L2, LLC, TLB, L2-TLB

○可推断

- 从缓存状态可反推信息
 - 一个页是否存在
 - 一个缓存块是否被访问

- 假设攻击者和被攻击者共享缓存
- 攻击者通过缓存反推关键信息
- 随机化后，攻击者很难寻找最小eviction set
- 通过随机化，阻止攻击者反推关键信息
- 通过随机化，阻止攻击者控制特定的缓存块状态

缓存随机化实现



为了挫败动态的驱逐集寻找算法，chipper所用的密钥会定期改变。

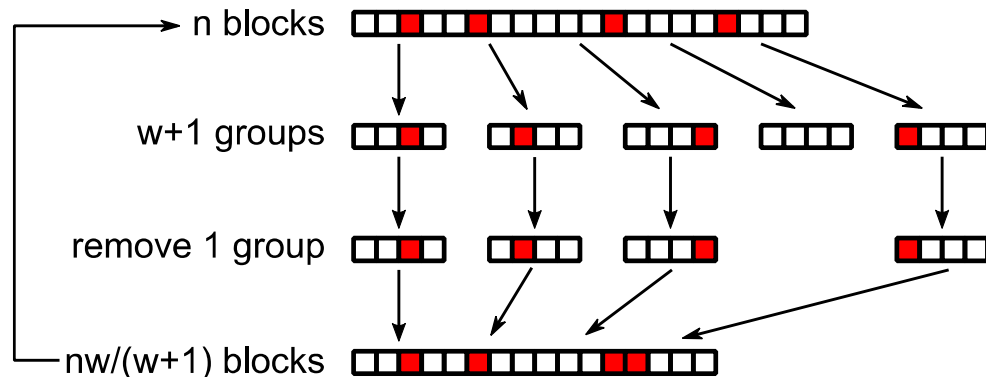
M. K. Qureshi. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. MICRO'18, 775-787

Algorithm 2: Prune with split: $\text{prune_split}(C, x, w, l)$

Input: C , candidate set; x , target address; w , number of ways; l , split parameter.

Output: Minimal eviction set for x .

```
1 function  $\text{prune\_split}(C, x, w, l)$ 
2   while  $|C| > w$  do
3      $\{G_1, \dots, G_l\} \leftarrow \text{split}(C, l)$ 
4     foreach  $G$  in  $\{G_1, \dots, G_l\}$  do
5       if  $\text{test}(C \setminus G, x)$  then
6          $C \leftarrow C \setminus G$ 
7       end
8     end
9   end
10  return  $C$ 
11 end
```

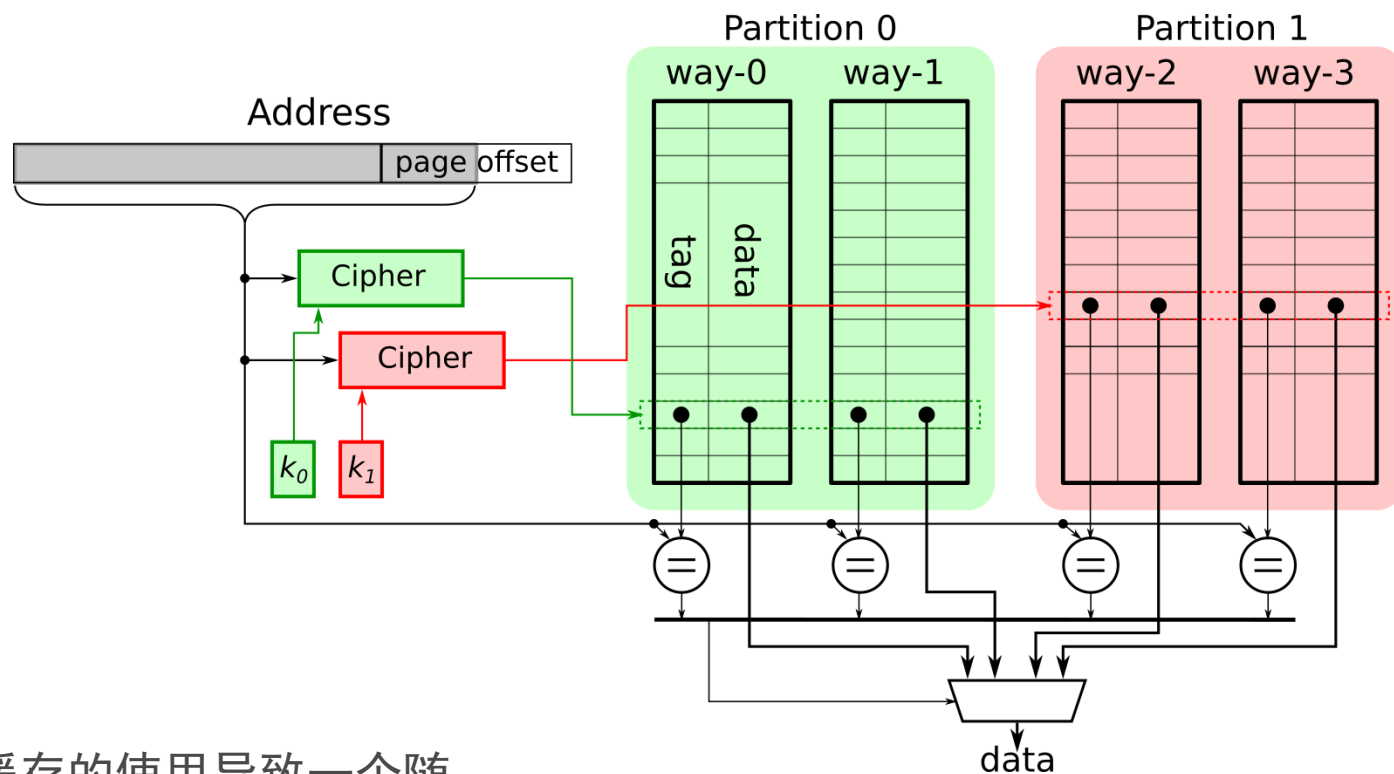


使用快速的驱逐集寻找算法将时间复杂度从 $O(N^2)$ 降低到了 $O(wN)$ 。

P. Vila, B. Köpf, J. F. Morales. Theory and Practice of Finding Eviction Sets. S&P'19, 39-54.

W. Song, P. Liu. Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC. RAID'19, 427-442.

随机化的skewed缓存



随机skewed缓存的使用导致一个随机地址和目标地址完全冲突的概率从原来的 $1/S$ 编程了 $1/(S^S)$ 。

M. K. Qureshi. New attacks and defense for encrypted-address cache. ISCA'19, 360-371.

M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, S. Mangard. ScatterCache: Thwarting cache attacks via cache set randomization. Security'19, 675-692.

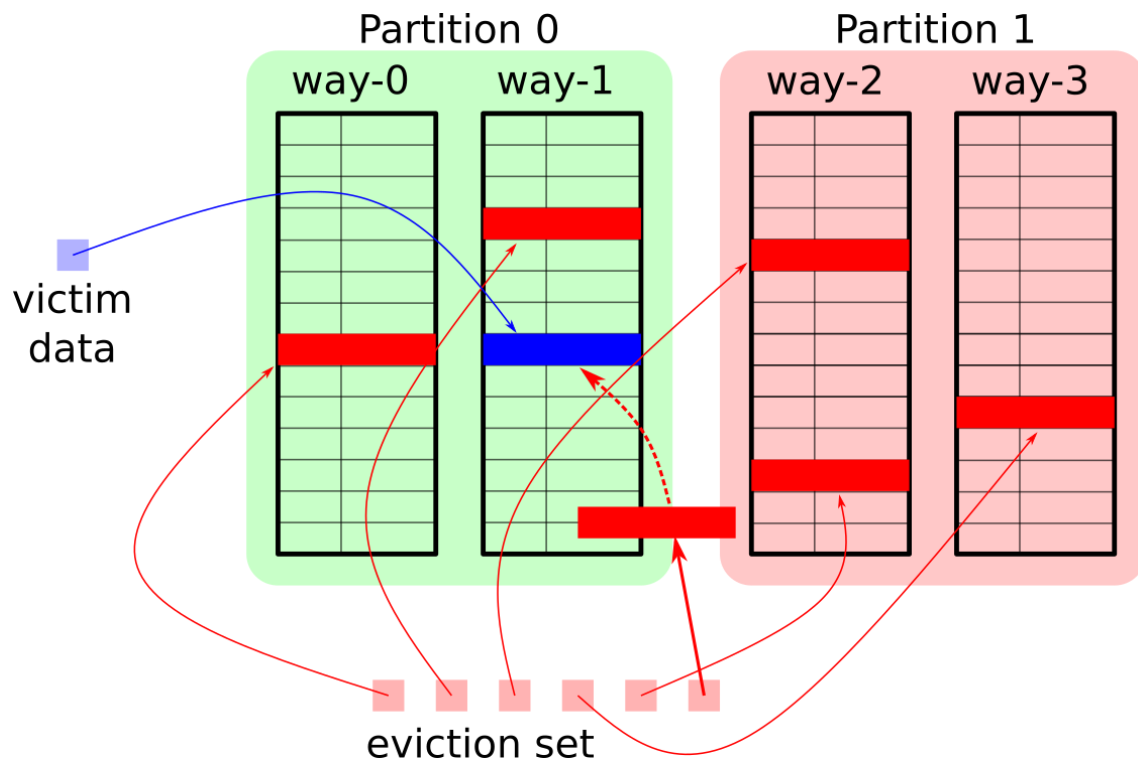
攻击随机化的skewed缓存

使用部分冲突的地址。

驱逐集不再是W个地址。

地址的数量越多，将目标地址踢出缓存的概率就越大。

但是，驱逐集似乎只能用一次。

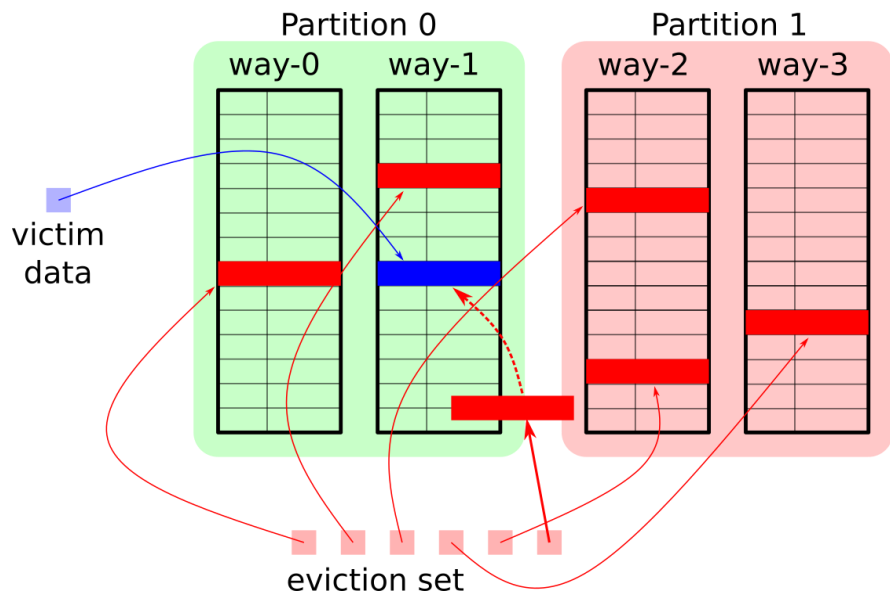


A. Purnal, L. Giner, D. Gruss, I. Verbauwhede. Systematic Analysis of Randomization-based Protected Cache Architectures. S&P'21.

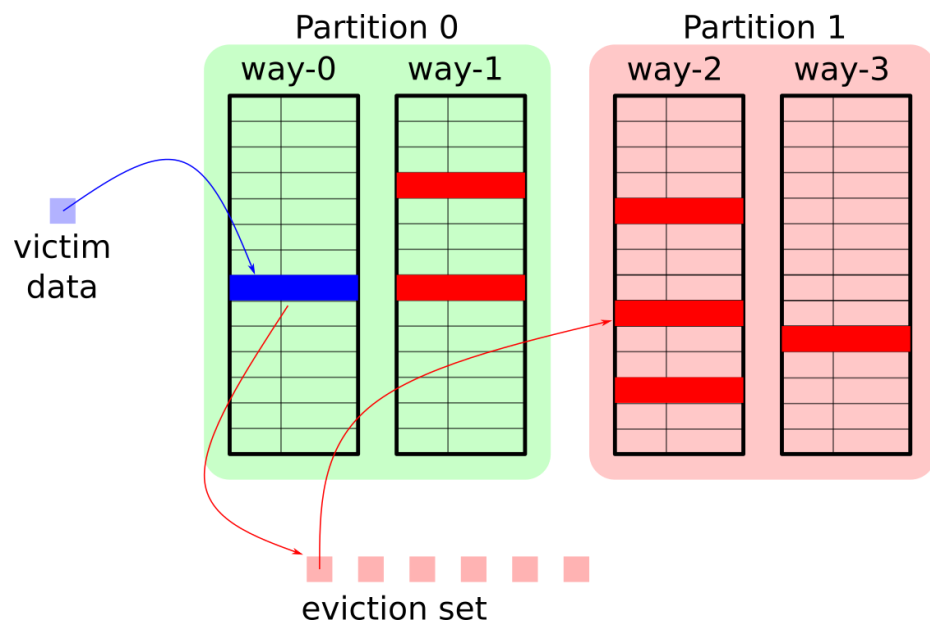
W. Song, B. Li, Z. Xue, Z. Li, W. Wang, P. Liu. Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It, S&P'21.

T. Bourgeat, J. Drean, Y. Yang, L. Tsai, J. Emer, M. Yan. CaSA: End-to-end Quantitative Security Analysis of Randomly Mapped Caches. MICRO'20, 1110-1123.

攻击随机化的skewed缓存



revisit the victim data



但是，驱逐集似乎只能用一次。

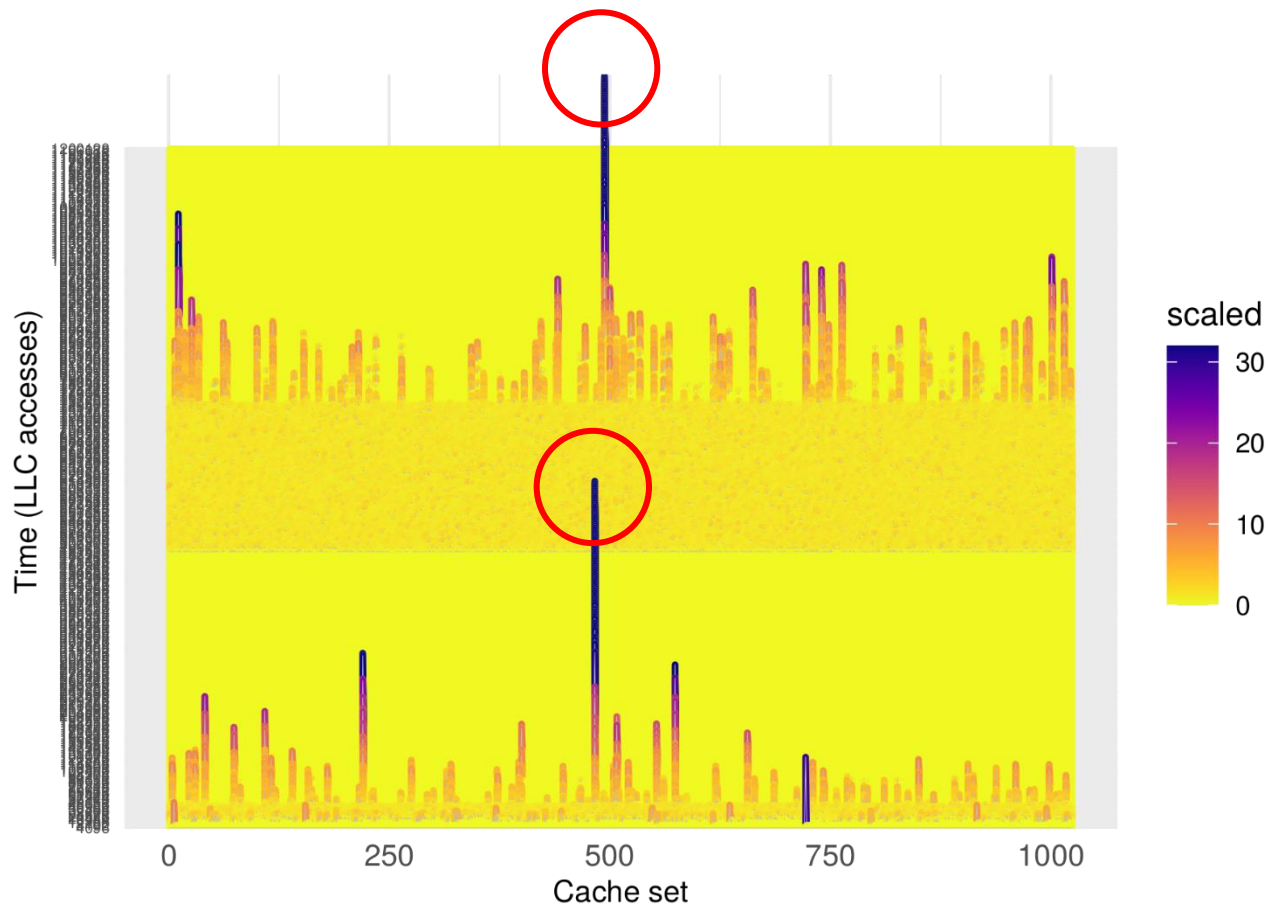
但是，我们可以用cflush把驱逐集刷掉。

A. Purnal, L. Giner, D. Gruss, I. Verbauwhede. Systematic Analysis of Randomization-based Protected Cache Architectures. S&P'21.

W. Song, B. Li, Z. Xue, Z. Li, W. Wang, P. Liu. Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It, S&P'21.

T. Bourgeat, J. Drean, Y. Yang, L. Tsai, J. Emer, M. Yan. CaSA: End-to-end Quantitative Security Analysis of Randomly Mapped Caches. MICRO'20, 1110-1123.

攻击随机化的skewed缓存



W. Song, B. Li, Z. Xue, Z. Li, W. Wang, P. Liu. Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It, S&P'21.

内容概要

- 缓存结构简述
 - 缓存结构、虚实地址转换、缓存一致性
- 针对L1的缓存侧信道
 - 基本攻击
 - Prime+Probe、Flush+Reload
- 针对LLC的缓存侧信道
 - Complex addressing、构造eviction set
- 防御措施
 - 缓存隔离
 - 缓存随机化

- 如何攻击L1-I 指令缓存。
- 如果LLC不是inclusive的会发生什么？
- 不同置换算法对攻击的影响。

- 如何攻击L1-I 指令缓存。
 - 构造代码做为eviction set
- 如果LLC不是inclusive的会发生什么？
 - 攻击可能不成功
- 不同置换算法对攻击的影响。
 - 增加噪声，加大eviction set长度