CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017

Lecture 8b

- Slide deck originally based on some material by Chenglu during ECE 6095 Spring 2017 on Secure Computation and Storage, a precursor to this course

Advanced Power Side Channel
Cache Side Channel Attacks
DMA Attacks

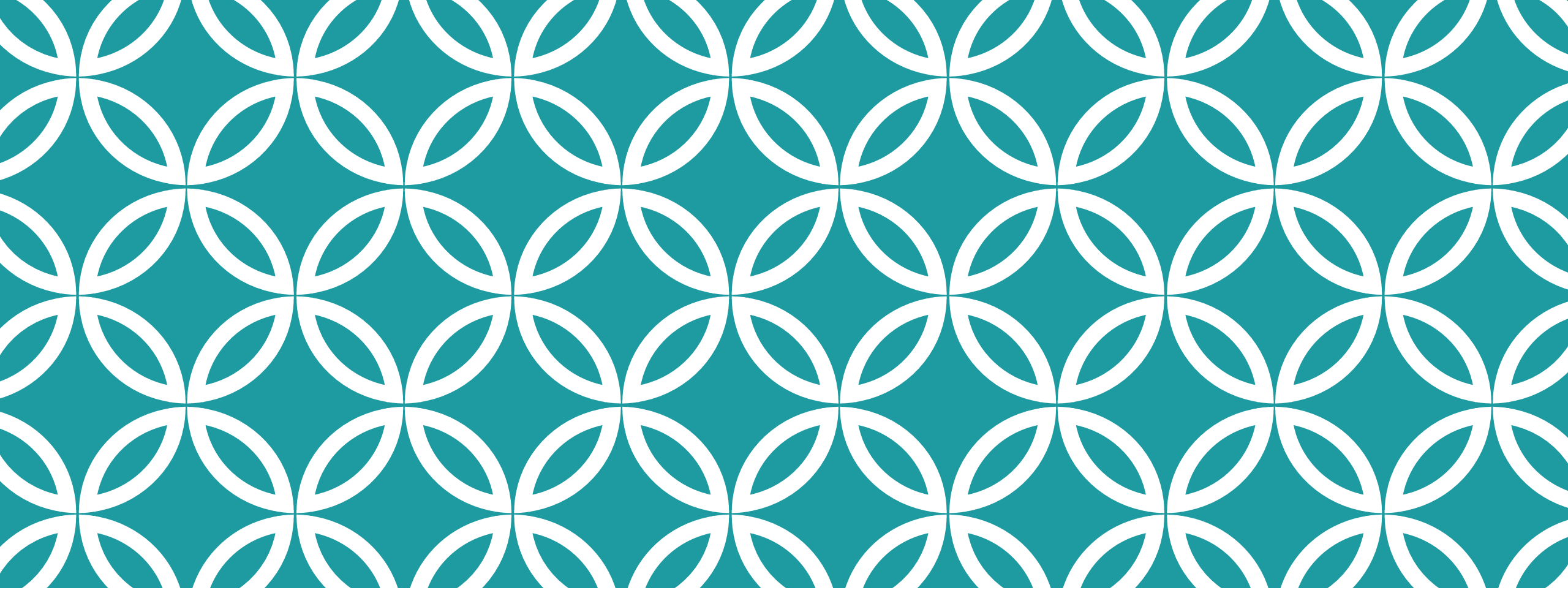
Marten van Dijk
Syed Kamran Haider, Chenglu Jin, Phuong Ha Nguyen

Department of Electrical & Computer Engineering
University of Connecticut

UCONN

Outline

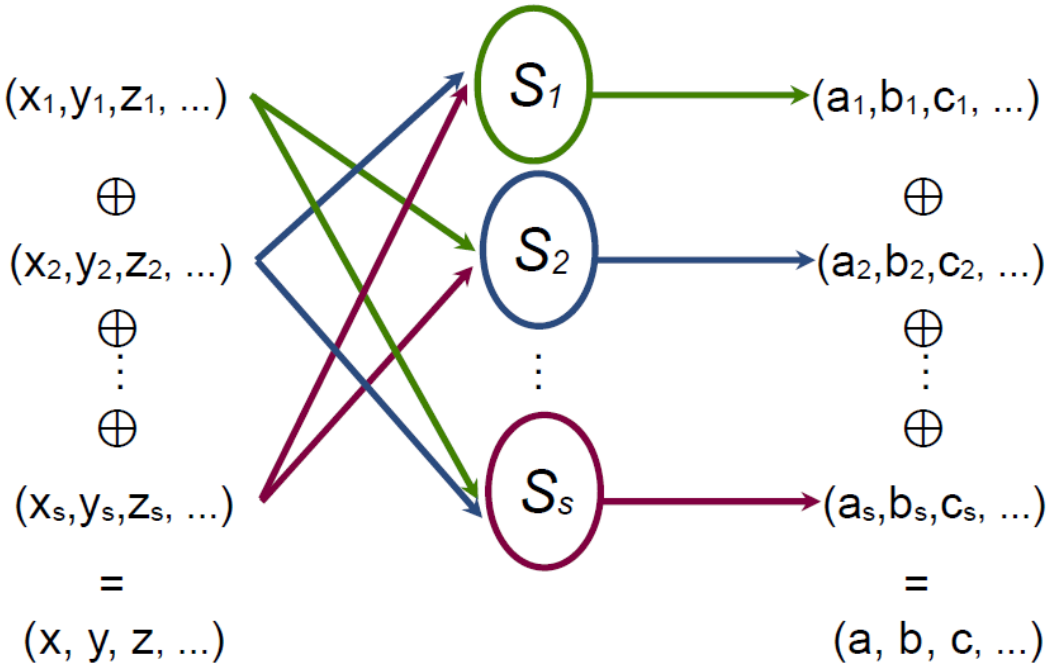
1. Advanced Power Side Channel Analysis
   1. High Order Attacks
   2. High Order Countermeasures
2. Cache Side Channel Attacks
   1. Attacks
   2. Countermeasures
3. DMA Attack



Advanced Power Side Channel

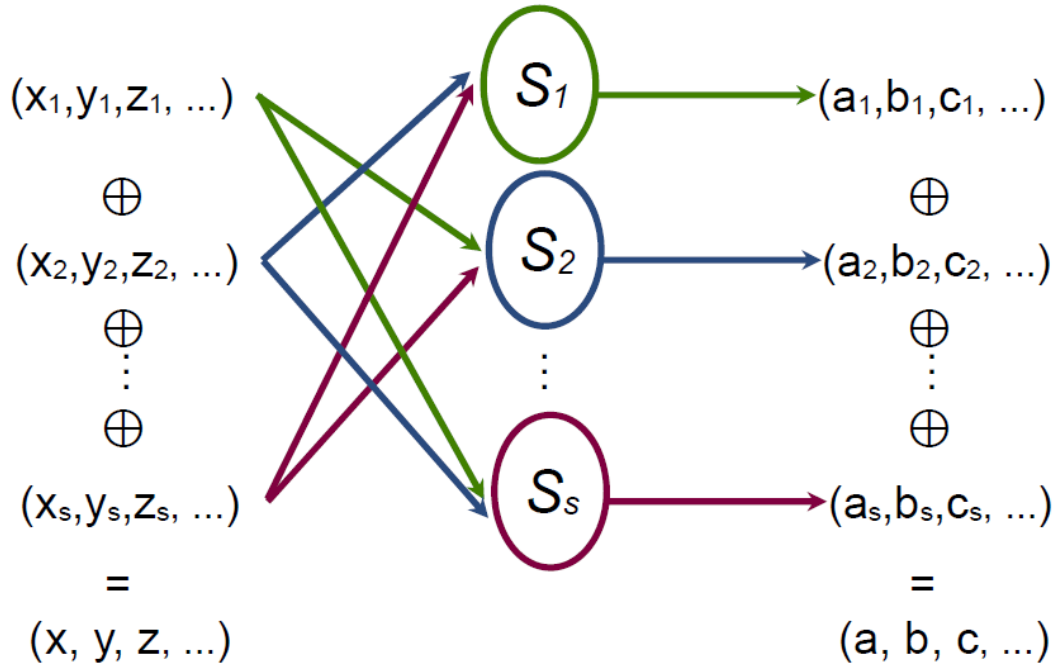
TI revisit

- Threshold Implementation offers provable security for one level of non-linear function even with glitches.
- If we share one bit information into 3 shares, then each non-linear operation will take at most two shares. This is called first-order TI, which is provably secure against first order attacks.
- Intuition: No single wire carries any unmasked (unrandomized) information.
- The order of attack is defined to be the number of intermediate values that attacker predict in the attacks.



High-order Attacks

- If several intermediate values are used to formulate the hypotheses, then the corresponding DPA attacks are called *higher-order* DPA attacks.
- For example, if I can combine the leakage of S1 and S2, then it is called second-order attack.



Second-order Attack

- A second-order attack simply applies a first-order attack to the **preprocessed** traces.
- Suppose that we are attacking a masking scheme, which means all the information is shared in two shares. U = Um XOR m

1. Choose two intermediate values *Um and m* as attack target. Since mask m is a random number, we do not know Um or m, but we can calculate a hypothetical value of U = Um XOR m.
2. Record the power traces and we do the preprocessing (discussed later).
3. Map *U* to hypothetical power consumption values *h*.
   - Note that we can calculate the value of U without having to know the mask!
4. Compare the hypothetical power consumption with the preprocessed traces by correlation coefficient.

Prerprocessing of traces

- Use the example on previous slide about masking scheme: U = Um XOR m
- If the computations of Um and m are in different clock cycles, we need to combine the sample points in different cycles. (Usually happens in software implementations)
- If the computations of Um and m are in the same clock cycle, we need to combine the sample points in the same cycle. (Usually happens in hardware implementations)
- Several different ways of preprocessing have been proposed: let two points be p0 and p1
  - $pre(p0, p1) = p0 \cdot p1$
  - $pre(p0, p1) = |p0 - p1|$
  - $pre(p0, p1) = (p0 + p1)^2$
  - $pre(p0, p1) = p0 + p1$

High-order attacks

- The methodology of second-order attack can be extended to even higher order.
- So how can we get the protection against high order attack?

High-order Countermeasures

- d-th order TI
  - d-th order non-completeness: any combination of up to d component functions y_i of Y must be independent of at least one input share.

Theorem *There always exist a d^{th}-order TI of a function of degree t that requires $s_{in} \geq t \times d + 1$ input and $s_{out} \geq \binom{s_{in}}{t}$ output shares.*

  - Example of second order TI of function Y(a,b,c) = 1 + a + bc
  - Any combination of two y_i is independent of at least one input share.

$$y_1 = 1 + a_2 + b_2c_2 + b_1c_2 + b_2c_1 \qquad y_2 = a_3 + b_3c_3 + b_1c_3 + b_3c_1$$

$$y_3 = a_4 + b_4c_4 + b_1c_4 + b_4c_1 \qquad y_4 = a_1 + b_1c_1 + b_1c_5 + b_5c_1$$

$$y_5 = b_2c_3 + b_3c_2 \qquad y_6 = b_2c_4 + b_4c_2$$

$$y_7 = a_5 + b_5c_5 + b_2c_5 + b_5c_2 \qquad y_8 = b_3c_4 + b_4c_3$$

$$y_9 = b_3c_5 + b_5c_3 \qquad y_{10} = b_4c_5 + b_5c_4\,.$$

Here, 5 input shares and 10 output shares.

Can we do better?

Domain Oriented Masking

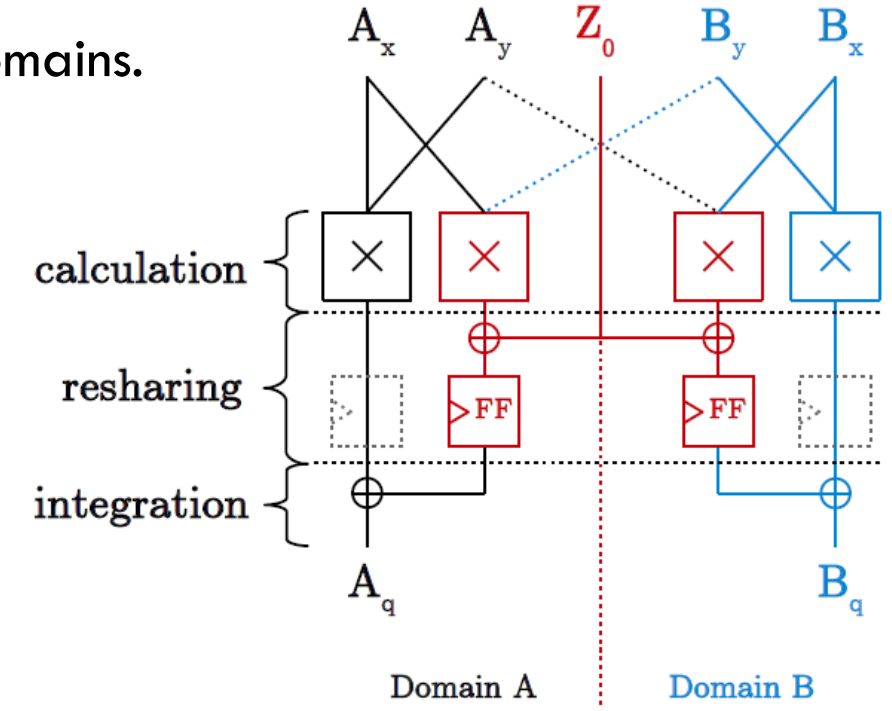
- Requires d+1 input/output shares for d-th order protection for functions with any degrees.
- Divide the computation in different domains. For d-th order protection, we need d+1 domains.
- Share every bit information into different domains.

$$\underbrace{A_q}_{Q_0} = \underbrace{\boldsymbol{A_x A_y}}_{t_{0,0}} + \underbrace{(A_x B_y + Z_0)}_{t_{0,1}}$$

$$\underbrace{B_q}_{Q_1} = \underbrace{(B_x A_y + Z_0)}_{t_{1,0}} + \underbrace{\boldsymbol{B_x B_y}}_{t_{1,1}}$$

Inner Domain Terms

Cross Domain Terms



First order secure AND gate.

Gross H, Mangard S, Korak T. "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order" CT-RSA'17 (Feb, 2017)

Domain Oriented Masking

- Second order protection:

$$
\begin{aligned}
\underbrace{\mathrm{A_q}}_{\mathrm{Q_0}} &= \underbrace{\boldsymbol{A_x A_y}}_{t_{0,0}} + \underbrace{(A_x B_y + Z_0)}_{t_{0,1}} + \underbrace{(A_x C_y + Z_1)}_{t_{0,2}} \\
\underbrace{\mathrm{B_q}}_{\mathrm{Q_1}} &= \underbrace{(B_x A_y + Z_0)}_{t_{1,0}} + \underbrace{\boldsymbol{B_x B_y}}_{t_{1,1}} + \underbrace{(B_x C_y + Z_2)}_{t_{1,2}} \\
\underbrace{\mathrm{C_q}}_{\mathrm{Q_2}} &= \underbrace{(C_x A_y + Z_1)}_{t_{2,0}} + \underbrace{(C_x B_y + Z_2)}_{t_{2,1}} + \underbrace{\boldsymbol{C_x C_y}}_{t_{2,2}}
\end{aligned}
$$

This methodology can be generalized to any d-th order protection with d+1 shares.

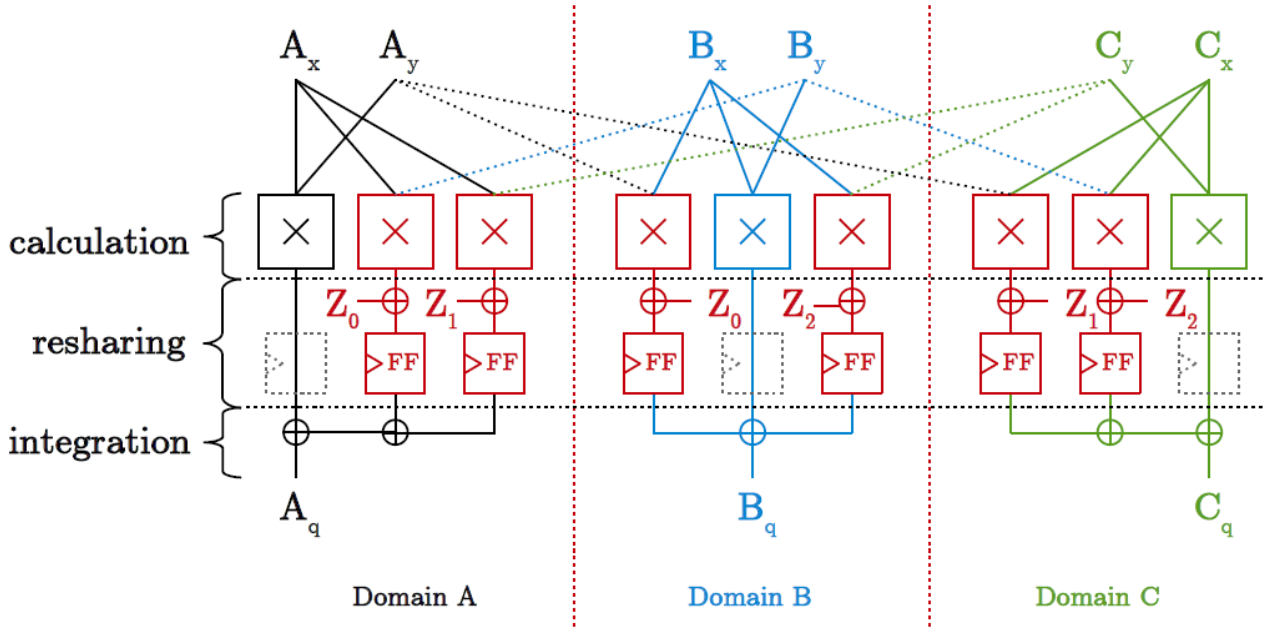
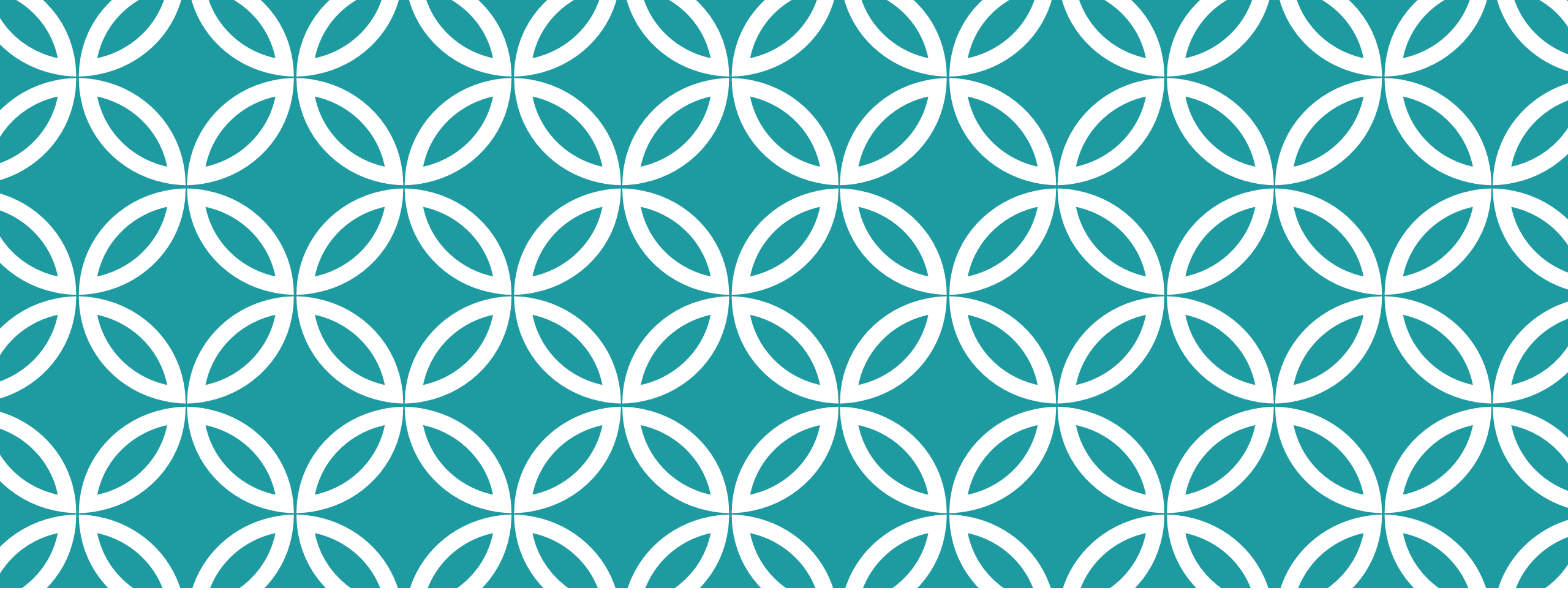


Fig. 2. Second-order secure AND gate

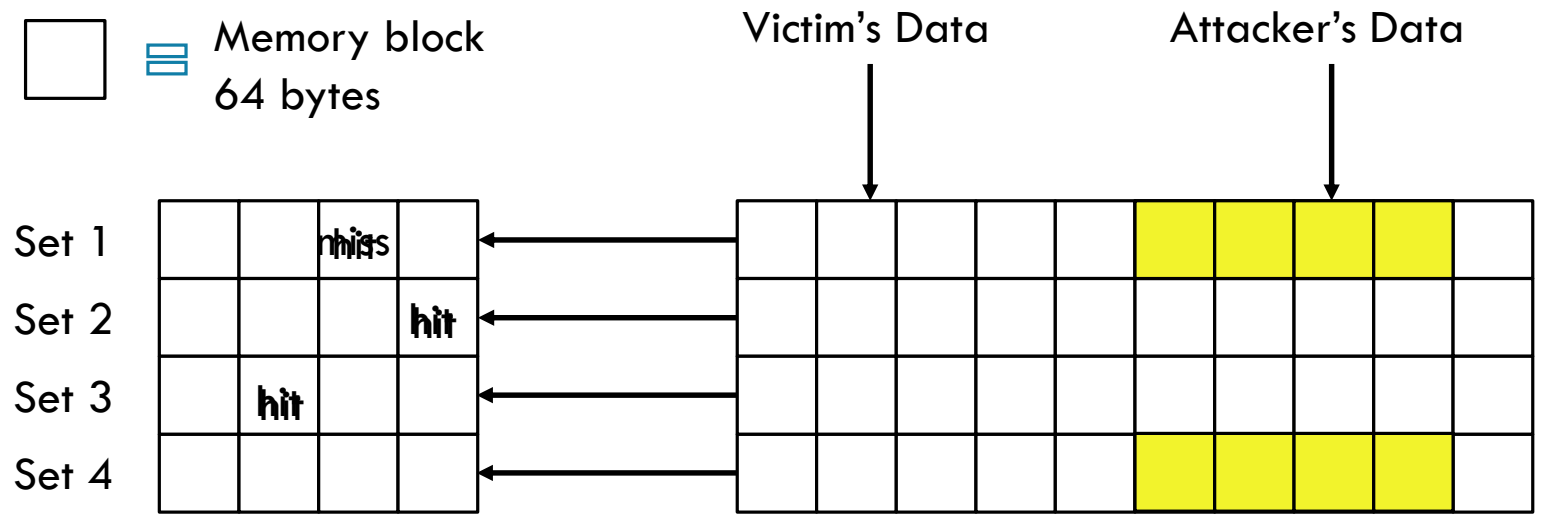


Cache Side Channel Attack

Cache side channel attacks

- Data present in caches can be accessed faster than from memory
- For multilevel caches, data accessed from L1 cache has lower latency than from an L2 cache
- The cache interference and time difference for the access patterns leaks information:
  - Certain memory contents exist in cache or not
  - Shows that data has been accessed recently
- This attack is useful to find keys for encryption process

Evict + Time Attack



Cache

The attacker wants to know if **0x1000**, which maps to cache **Set 1**, was accessed

- He triggers the encryption and times it.
- He evicts everything from **Set1**.
- He runs the encryption again and times it.
- It takes longer than step 1, he knows that the encryption process accessed **0x1000**.

Main Memory

The attacker wants to know if **0x4000**, which maps to cache **Set 4**, was accessed

- He triggers the encryption and times it.
- He evicts everything from **Set4**.
- He runs the encryption again and times it.
- It takes roughly the same time, he knows that the encryption process didn't access **0x4000**.

Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: the case of AES[M]//Topics in Cryptology–CT-RSA 2006. Springer Berlin Heidelberg, 2006: 1-20.

AnC Attack

- Evict + Time attack on MMU (Memory Management Unit) to break ASLR (Address Space Layout Randomization) completely.
- The memory management unit (MMU) of modern processors uses the cache hierarchy in order to improve the performance of page table walks.
- An EVICT+TIME cache attack can detect which locations in the page table pages are accessed during a page table walk performed by the MMU.
- For example, on the x86_64 architecture, this attack can find the offsets that are accessed by the MMU for each of the four page table pages.
- The attack, which is called ASLR⊕Cache (or AnC for short), first flushes part of the last level cache and then times the MMU's page table walk performed due to a memory access. This already finds cache lines of interest in the page table page.

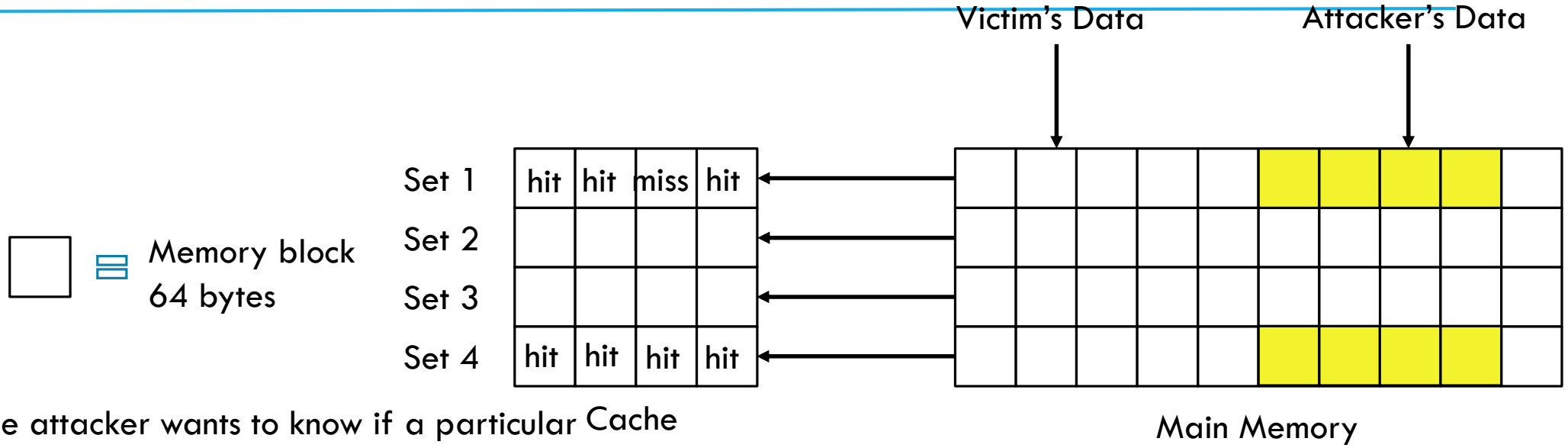
Gras B, Razavi K, Bosman E, et al. "ASLR on the Line: Practical Cache Attacks on the MMU", NDSS'17 (Feb. 2017)

Prime + probe technique

- Prime + probe technique consists of 3 stages
  - Prime stage : The attacker fills the cache with his own cache lines.
  - Victim accessing stage : The victim process runs
  - Probing stage : The attacker accesses the priming data again. If the victim process evicts the primed data, the reloading will incur cache miss.

Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: the case of AES[M]//Topics in Cryptology–CT-RSA 2006. Springer Berlin Heidelberg, 2006: 1-20.

Prime + probe technique



The attacker wants to know if a particular Cache address in cache **Set 1** was accessed

- He fills **Set1** with his data.
- He runs the victim process.
- He reloads all his data in **Set1**.
- It takes longer, he knows that the victim process accessed **Set1**.

The attacker wants to know if a particular address in cache **Set 4** was accessed

- He fills **Set4** with his data.
- He runs the victim process.
- He reloads all his data in **Set4**.
- It takes lesser time, he knows that the victim process didn't access **Set4**.

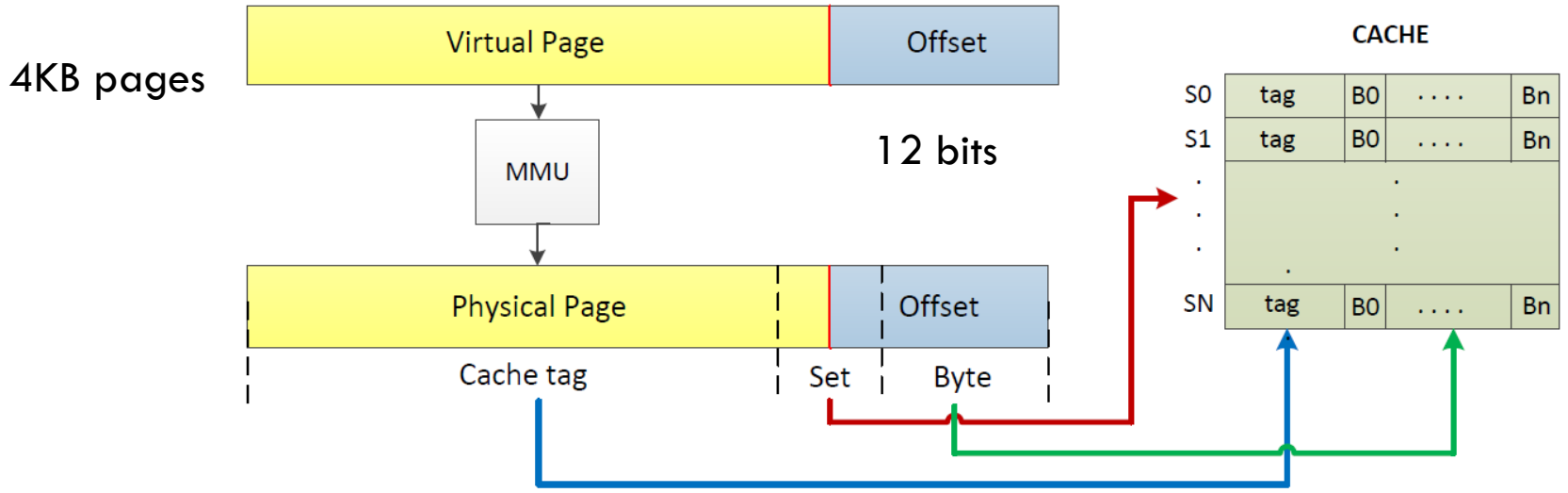
Taken from the presentation of "3D Integration: New Opportunities in Defense Against Cache-timing Side-channel Attacks" by Chongxi Bao and Ankur Srivastava on ICCD'15.

Lab 4

- In lab 4, you are required to finish prime and probe attack.
- In one application, you need to create two threads: one is attack thread, the other one is victim thread.
- Context switching is implemented by using pin tool to suspend the execution of one of the threads.
- The latency of a cache hit and a cache miss can be implemented as a delay function which delays for a few milliseconds.

Limitations

- Can only be applied in small caches (L1 caches)



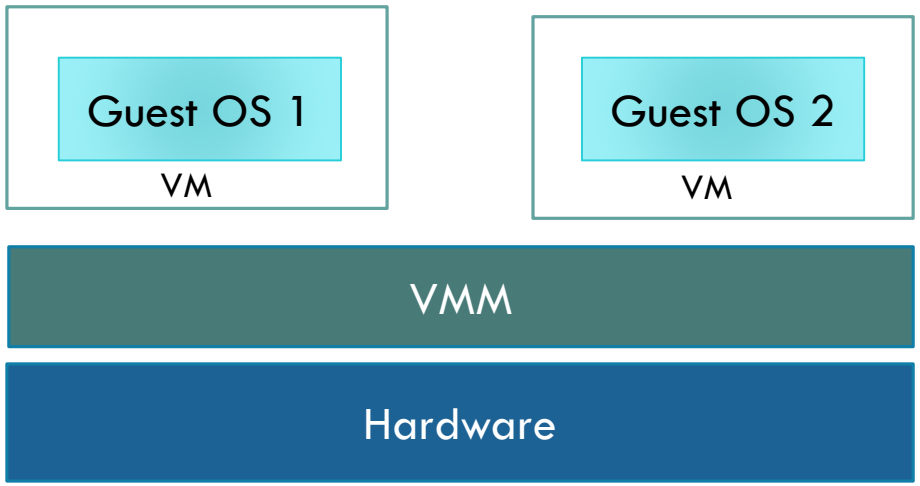
Cache line size = 64 bytes
Offset for cache = 6 bits

Cache index = 6 bits
at most to access 64 sets

- Since it is used in small caches its applicable to processes located in the same core

Practical Scenario

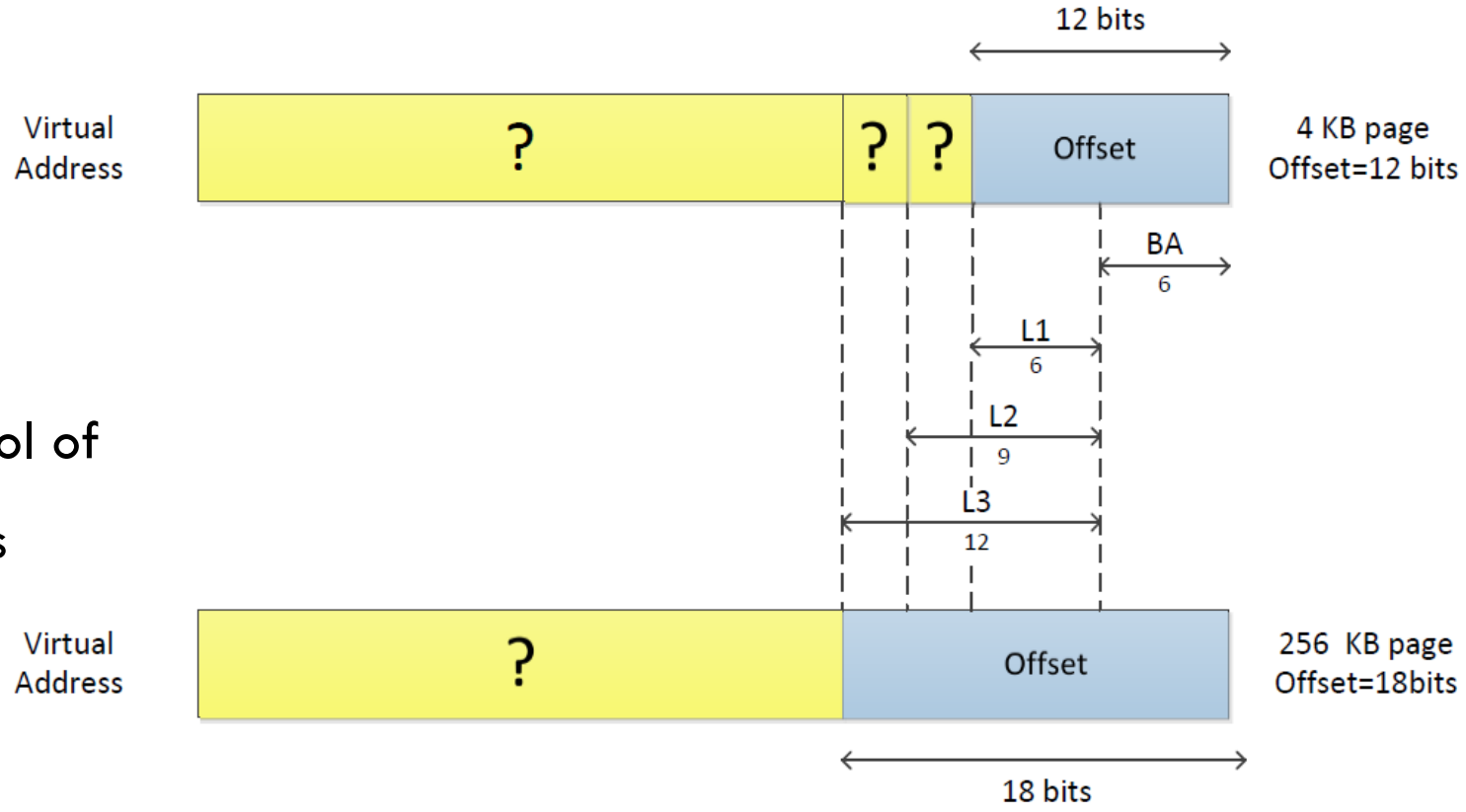
- In Cloud computing environment two users can share same hardware



- Users running on different cores share the last level cache

S$A attack (Shared Cache Attack)

- S$A attack is targeted towards the LLC
- Make use of huge size pages
- L1 – 64 sets
- L2 – 512 sets
- L3 – 4096 sets
- Takes advantage of the control of lower bits of the virtual address



Gorka Irazoqui, Thomas Eisenbarth and Berk Sunar, "S$A: A Shared Cache Attack that Works Across Cores and Defies VM Sandboxing—and its Application to AES", Oakland'15

Steps involved in S$A attack

1. Allocation of huge size pages
   - Spy process have access to huge pages using his administrator rights in guest OS
2. Prime desired set in last level cache
   - Attacker creates data that fills a set in the LLC and primes it
3. Reprime
   - Since LLCs are inclusive some sets in the upper level will also be filled
   - Evict data from the upper level caches
   - Reprime data to fill different set in LLC but same set in upper level cache

Steps involved in S$A attack

4. Victim process runs
   - Victim runs the target process
   - If monitored cache set is used, some of the primed lines will be evicted
   - Else all primes lines will reside in the LLC
5. Probe and measure
   - After execution of victim process, spy process probes the primed memory lines and measures the time to probe
   - If one or more lines have been evicted probe time will be higher
   - Shorter probe time if no lines were evicted

Flush + Reload Attack

- Flush one cache line and time the execution of reloading the value to figure out whether the victim program has access this cache line or not.
- Fine-grained: attack at cache line granularity.

Yarom Y, Falkner K. Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. USENIX Security 14

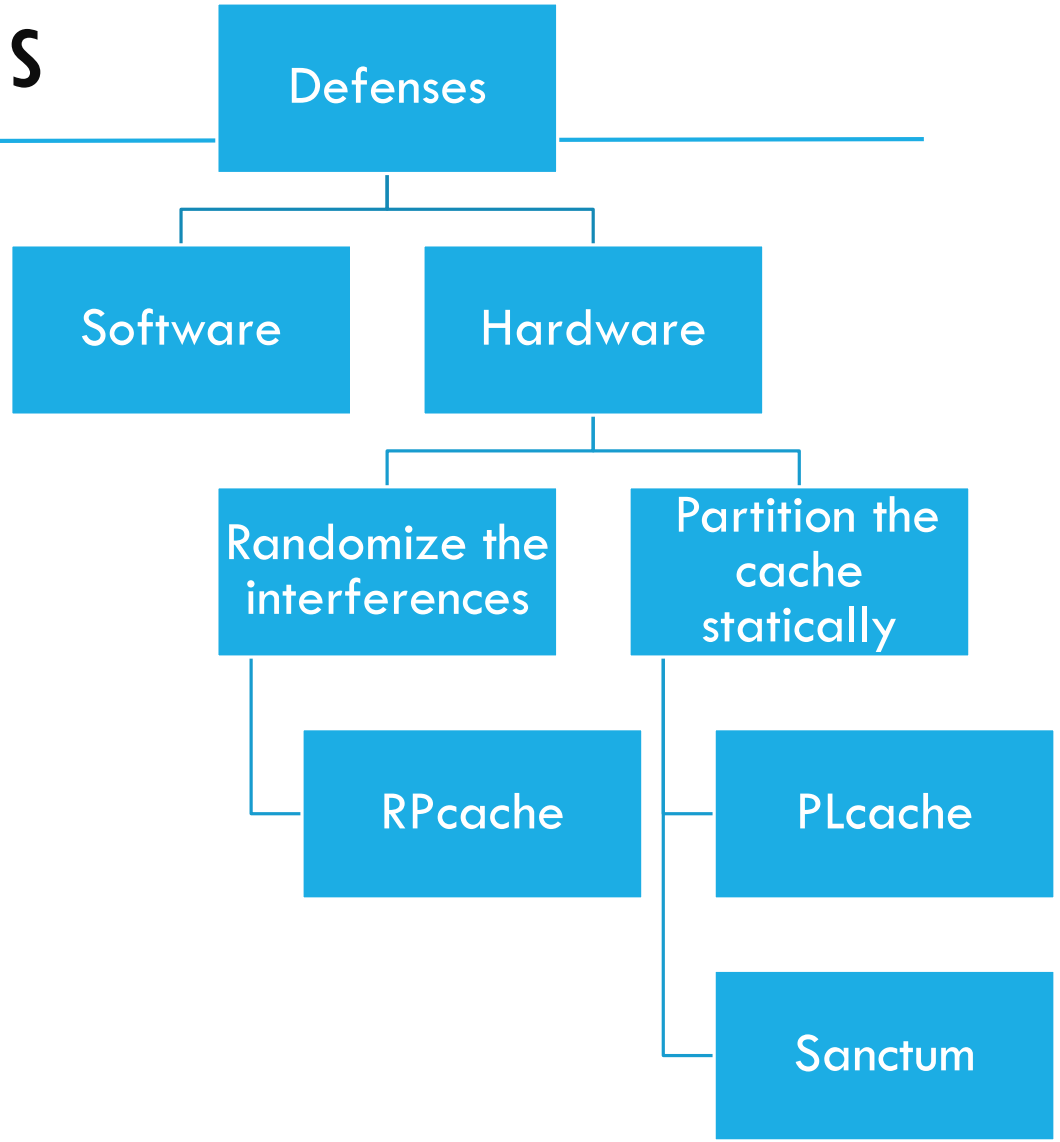
Flush + Flush Attack

- The same idea as Flush + Reload attack.
  - Problem: incur too much cache misses in reloading process, which may be used as a signature to detect cache side channel attack
- Flush + Flush attack exploits the execution time of Flush instruction to learn whether the Flush instruction hit the cache or not.
- Flush instruction can abort early in case of a cache miss. In case of a cache hit, it has to trigger eviction on all local caches, so it would take longer.
- Attack at cache line granularity, but less accuracy than Flush + Reload
- More stealthier, because incur fewer cache misses

Gruss D, Maurice C, Wagner K. Flush+ Flush: A Stealthier Last-Level Cache Attack. arXiv preprint arXiv:2015

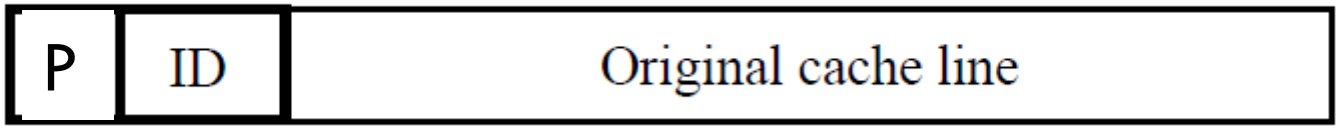
Defenses

- Cache interferences are the root causes of cache side channel attacks.
- Software-based approaches are all attack specific and algorithm specific.
- Hardware-based approaches:
  - Randomize the cache interferences -> no information leakage through interference.
  - Partition the cache statically -> no cache interferences.



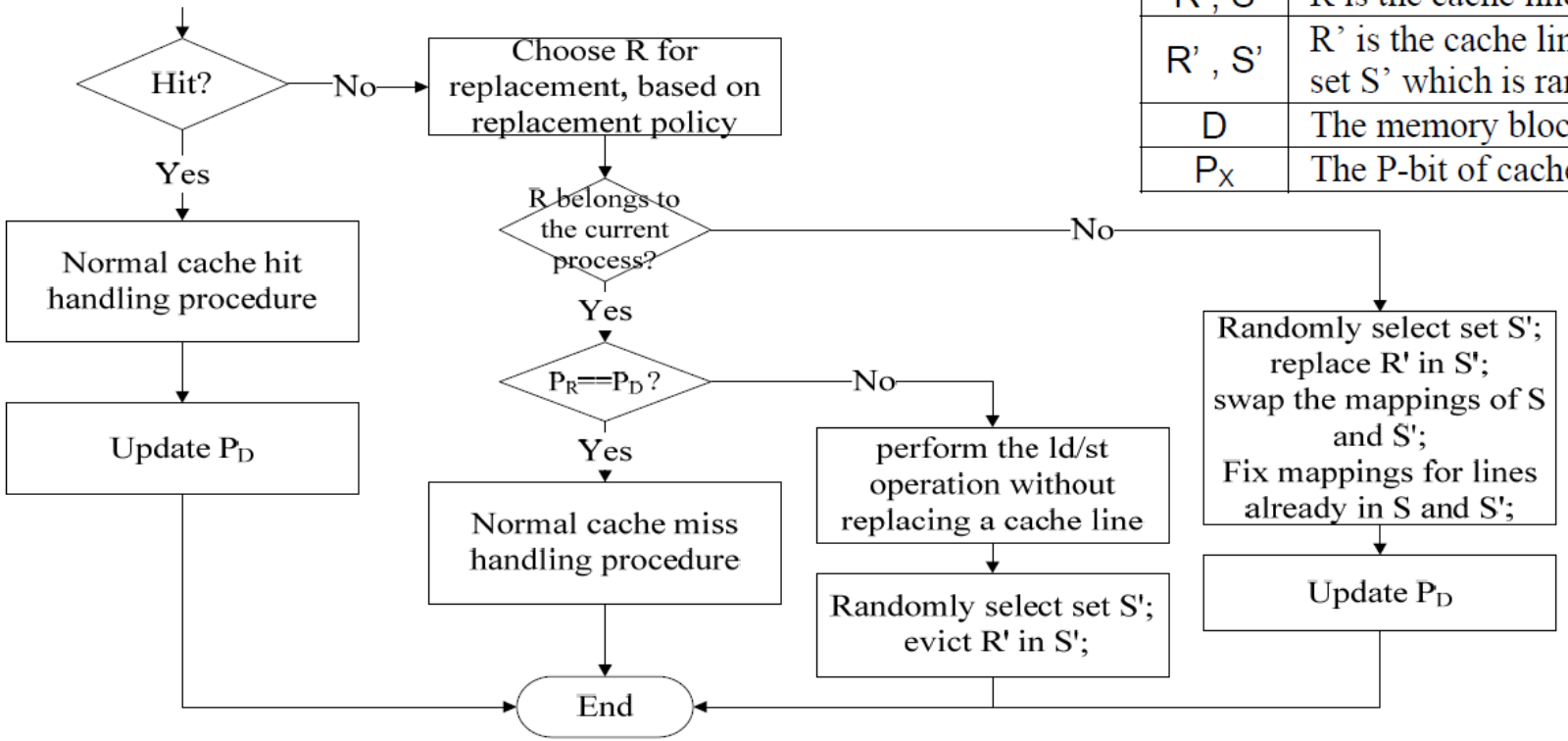
RPcache (Random Permutation Cache)

- Randomizes cache-memory mapping, when a cache interference occurs, so no useful information about which cache line was evicted can be inferred.



Zhenghong Wang and Ruby B. Lee “New cache designs for thwarting software cache-based side channel attacks”, ISCA 2007.

Cache access handling procedure



Name	Description
R , S	R is the cache line being replaced in cache set S.
R' , S'	R' is the cache line being replaced in another cache set S' which is randomly selected.
D	The memory block being fetched into the cache.
P_X	The P-bit of cache line X, e.g., of R, R' or D.

Figure 6. Cache access handling procedure for RPcache

Logical view

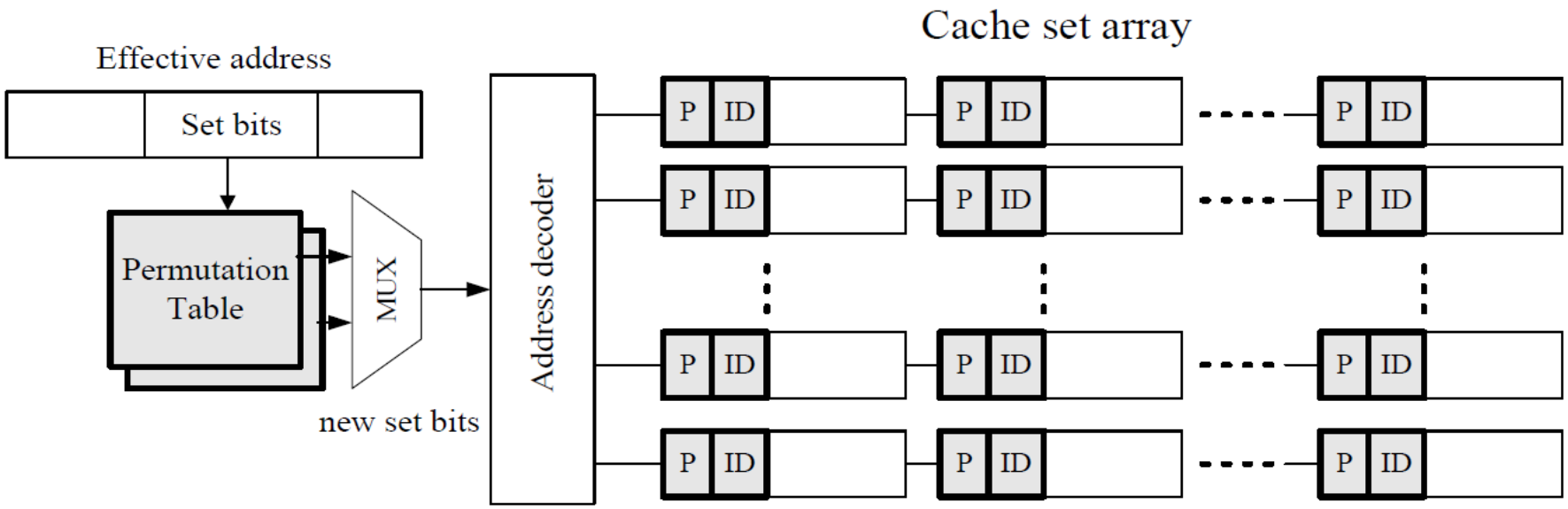
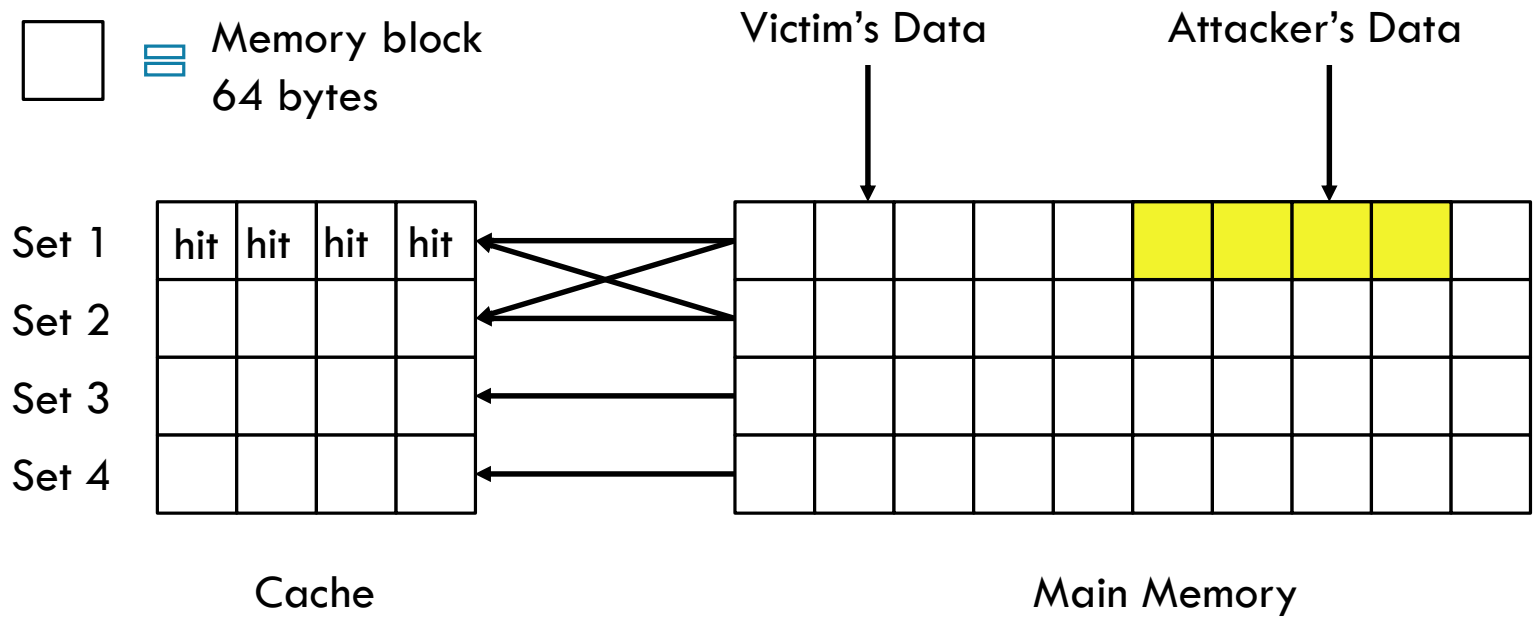


Figure 5. A logical view of the RPcache

Table 3. Timing and Power Estimation of RPcache

RPcache	16K 2way	32K 2way	16K 4way	32K 4way
Access time(ns)	1.225 (+2.1%)	1.331 (+1.7%)	1.293 (+1.1%)	1.344 (+3.3%)
Power (nj)	1.205 (+8.6%)	1.282 (+1.3%)	1.792 (+6.1%)	1.906 (+2.1%)

Example of RPCache



1. Attacker fills set 1.
2. Attacker runs the encryption process.
3. Victim's data maps to set 2 instead of set 1, and the mapping is swapped.
4. Attacker tries to access his data, and the mapping is swapped randomly again, so the hit rate of attacker's data does not infer any memory access of victim.

PLCache (Partition-Locked Cache)

- A process is able to lock the cache lines in the cache, so the cache will not be evicted by the data of other processes.

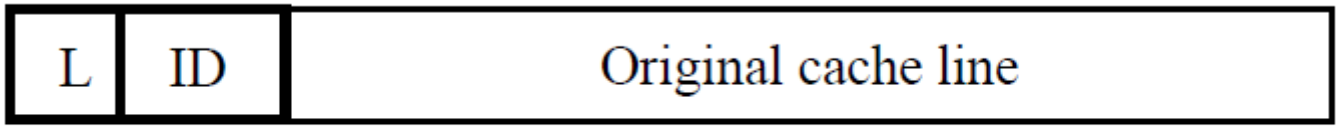


Figure 3. A cache line of the PLcache

Zhenghong Wang and Ruby B. Lee "New cache designs for thwarting software cache-based side channel attacks", ISCA 2007.

Cache access handling procedure

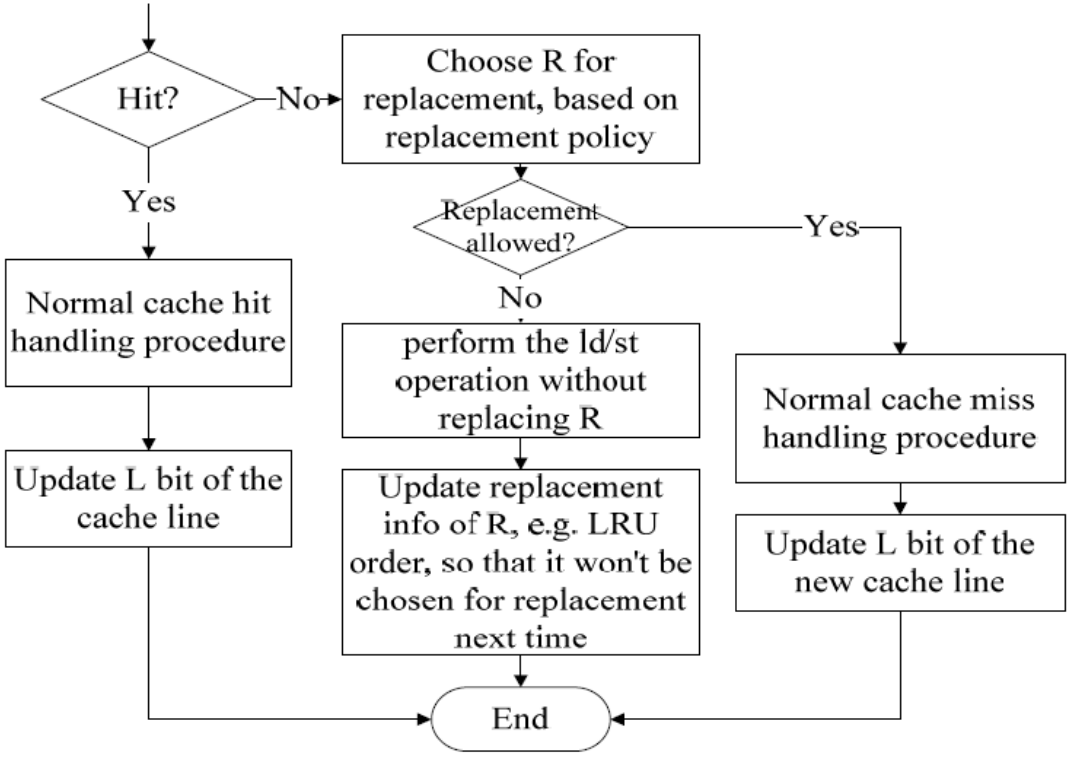


Figure 4. Access handling procedure for PLcache

Performance Evaluation

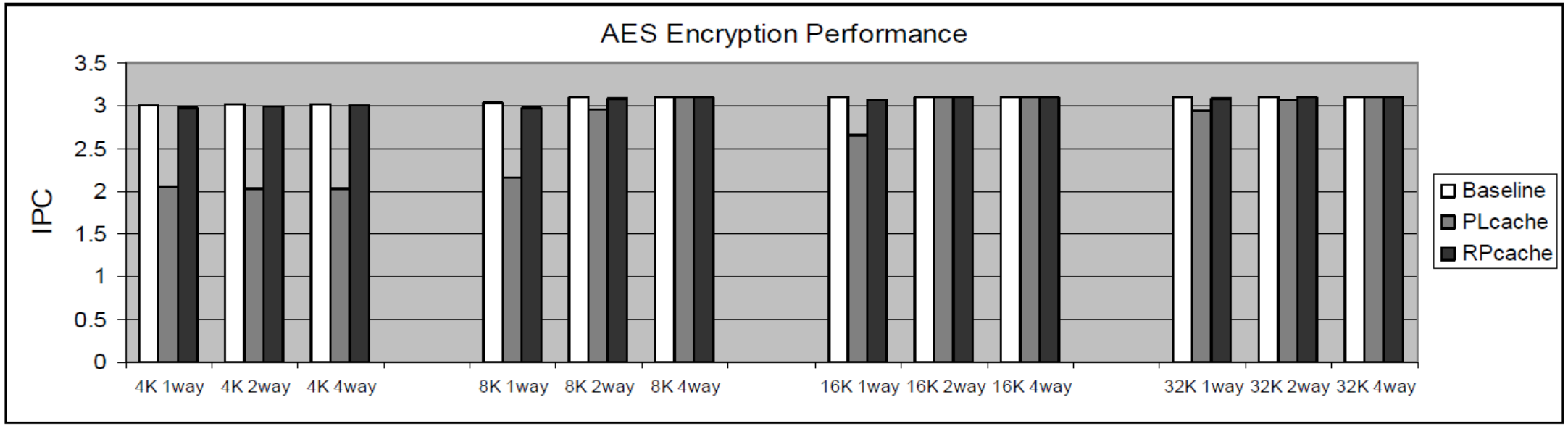


Figure 9. Performance comparison of AES code

RPCache: The performance impact caused by the random cache evictions in RPcache is negligible: worst case 1.7% (on 4K directed-mapped cache) and 0.3% on average.

PLCache: When the size of the protected memory (5KB) is larger than the cache capacity (4KB cache), the performance is always bad because all cache lines are locked. Set-associativity affects performance as well, direct-mapped cache has ~30% overhead.

Attack on PL Cache

- PL cache can protect the cache lines from evicting from the cache by other processes, but it does not prevent the cache access when we start loading the victim's cache line.
- Evict + Time does not work any more.
- Prime + probe still works.
- Flush + Reload still works.
- Flush + Flush still works

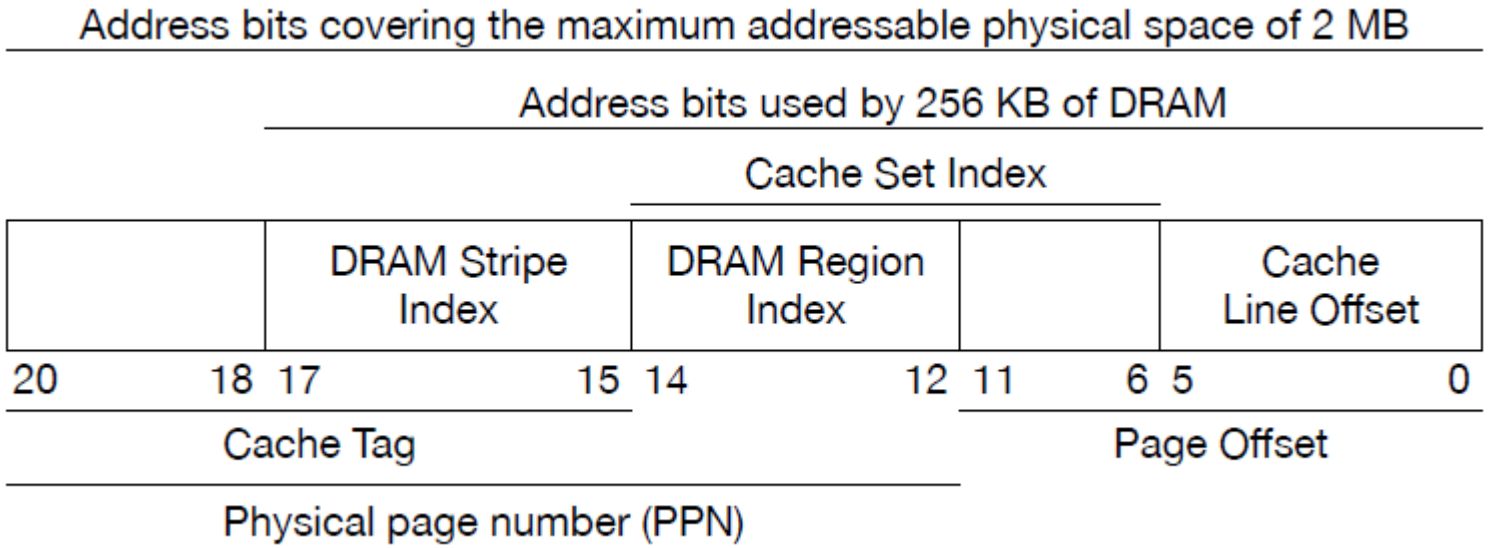
Kong J, Aciicmez O, Seifert J P, et al. Deconstructing new cache designs for thwarting software cache-based side channel attacks, ACM workshop on Computer security architectures. 2008

Sanctum

- Sanctum offers **strong provable isolation of software modules** running concurrently and sharing resources, but protects against the attacks that infer private information from a program's memory access patterns, including **cache side channel attacks**.
- Like SGX, Sanctum isolates the software inside an **enclave** from any other software on the system, including privileged system software.

Costan V, Lebedev I, Devadas S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation[J].

Static DRAM/LLC Partitioning



- Addresses in a DRAM region do not collide in the last level cache with addresses from any other DRAM region. So the OS can place two different applications in two different DRAM regions, then the cache interference in the last level cache is eliminated.
- For high level caches, Sanctum flushes them whenever a core jumps between enclave and non-enclave code.

Cache address shifter

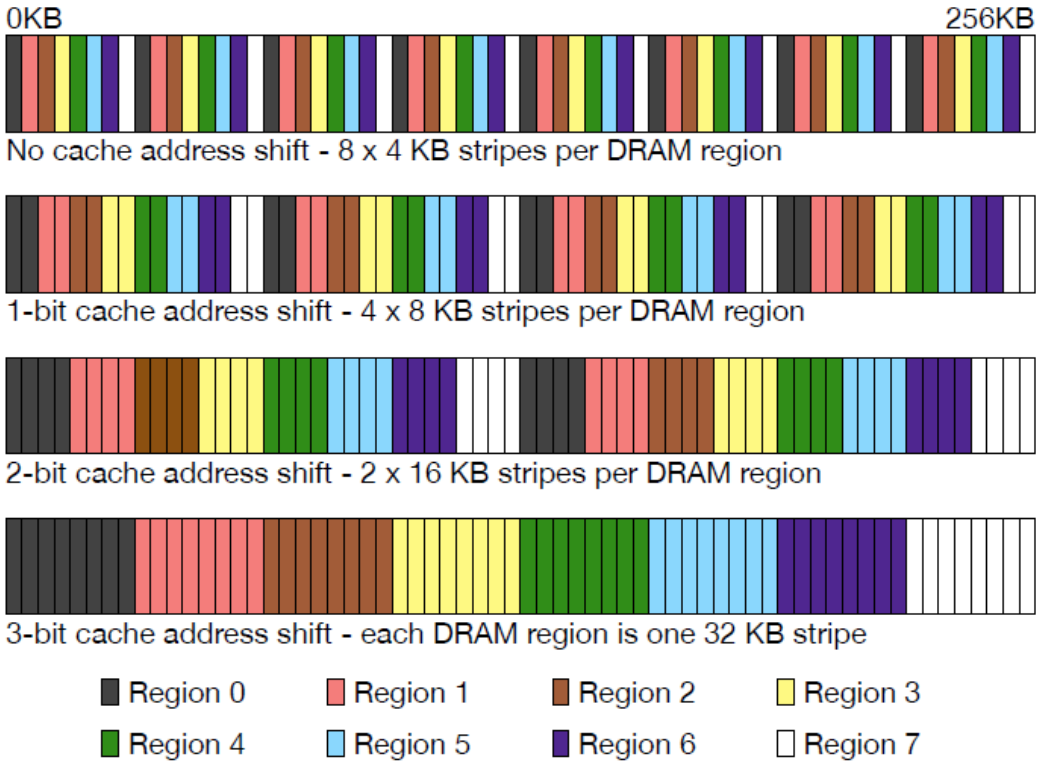
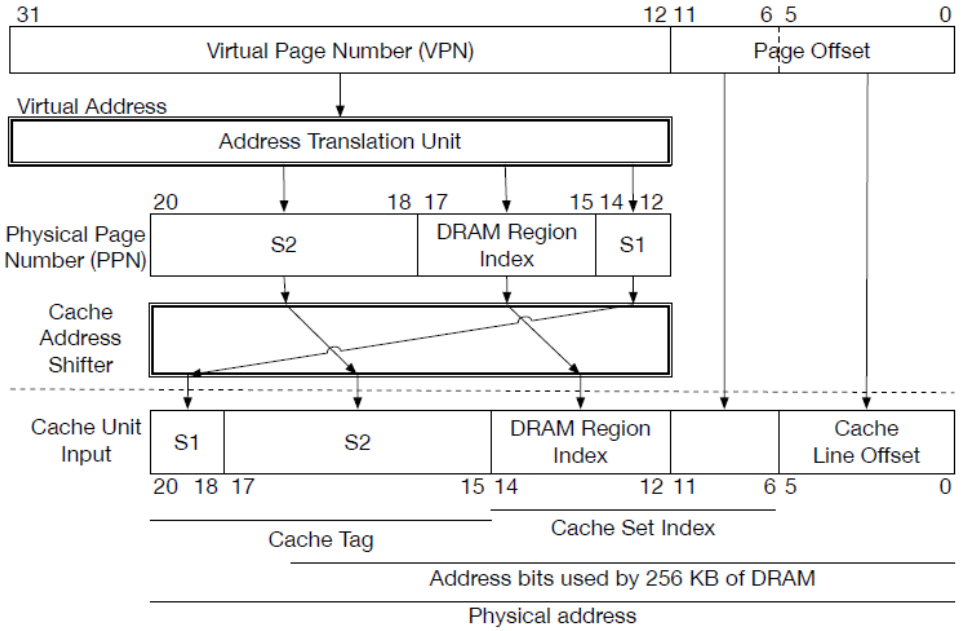


Figure 5: Cache address shifting makes DRAM regions contiguous

The fragmentation of DRAM regions makes it difficult for the OS to allocate contiguous DRAM buffers, which are essential to the efficient DMA transfers used by high performance devices.

Shifting the physical page number by 3 bits yields contiguous DRAM regions.



Performance Evaluation

Sanctum: Largest overhead is 4%, and average is 1.9% on an insecure baseline.

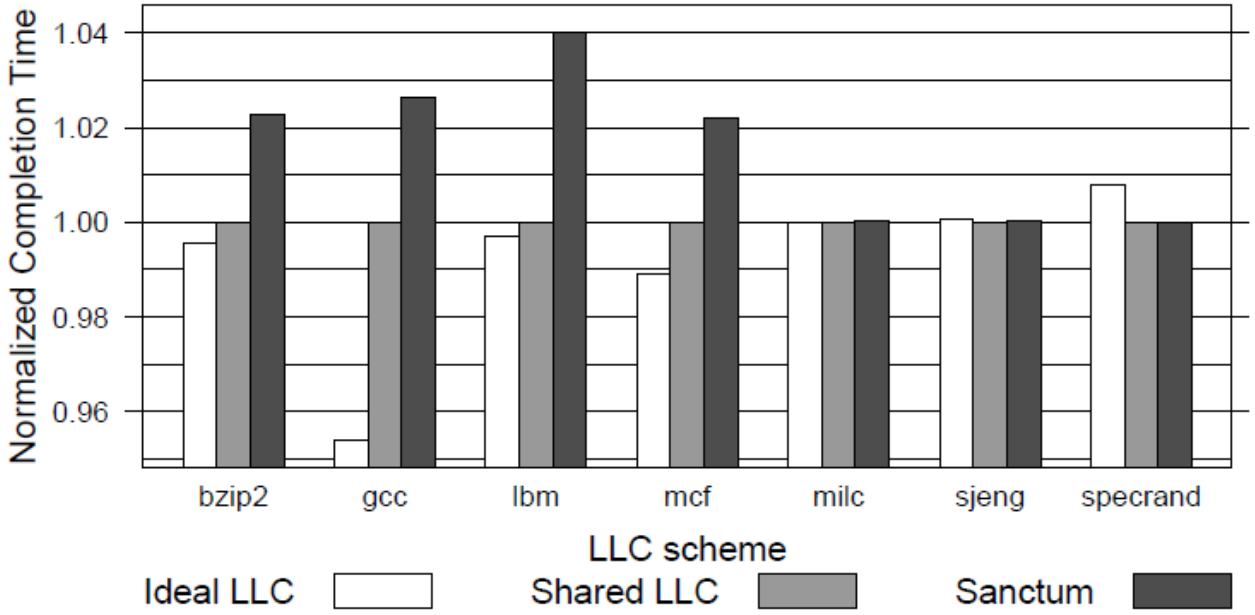
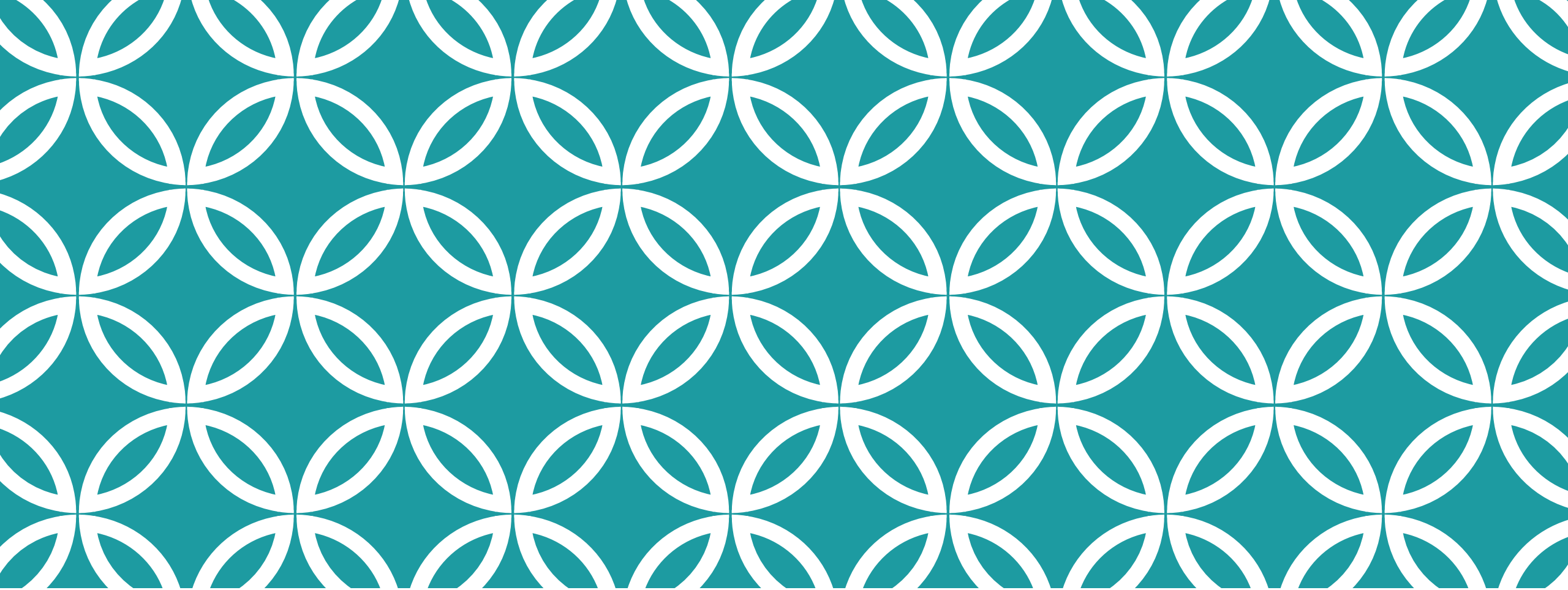


Figure 16: Sanctum's enclave overheads for one core utilizing 1/4 of the LLC compared against an idealized baseline (non-enclaved app using the entire LLC), and against a representative baseline (non-enclaved app sharing the LLC with concurrent instances)



DMA Attack

DMA Attack

- By dumping memory content very frequently, we are able to observe memory write patterns.
- Some algorithms can leak secret through memory write patterns.
- For example, RSA Montgomery ladder.
- It is designed to prevent simple power analysis or coarse grained timing attack. But it introduces another attack surface, which is the privacy leakage through write access patterns. This is what we should pay attention, when we are designing a system.

$x_1=x;\ x_2=x^2$
for i=k-2 to 0 do
  If $n_i=0$ then
    $x_2=x_1*x_2;\ x_1=x_1^2$
  else
    $x_1=x_1*x_2;\ x_2=x_2^2$
return x_1

References [1]

1. http://ca.olin.edu/2005/fpga_dsp/fpga.html
2. http://ca.olin.edu/2005/fpga_dsp/fpga.html
3. https://upload.wikimedia.org/wikipedia/commons/5/5c/Side_channel_attack.png
4. https://en.wikipedia.org/wiki/Power_analysis
5. http://people.eku.edu/styere/Encrypt/JS-DES.html
6. http://www.tutorialspoint.com/cryptography/data_encryption_standard.htm
7. Paul Kocher, Joshua Jaff and Benjamin Jun: Differential Power Analysis
8. Farinaz Koushanfar: Physical Security and Side Channel Attacks (presentation)
9. http://www.embedded.com/print/4199399
10. Differential Power Analysis presented by Italo Dacoseta (presentation)

References [2]

11. Differential Power Analysis Side Channel Attacks in Cryptography (Bachelor Thesis of William Hnath)
12. http://pubs.sciepub.com/iscf/3/1/1/
13. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
14. SIDE-CHANNEL ATTACKS ON HARDWARE IMPLEMENTATIONS OF CRYPTOGRAPHIC ALGORITHMS by Sıddıka Berna Ors Yal¸ cın (presentation)
15. Side Chanel Attack and Countermeasures for Embedded System by Job de Haas (presentation)
16. Overview of Countermeasures against Implementation Attack by Macel Medwed (presentation)
17. SPA and DPA attacks. Pascal Paillier
18. Lecture7 –More on Attacks Farinaz Koushanfar
19. Power Analysis Attacks. Elisabeth Oswald
20. Power Analysis – an overview. Benedikt Gierlichs

References [3]

21. https://cryptography.gmu.edu/fobos/
22. https://en.wikipedia.org/wiki/Side-channel_attack
23. Paul Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. Crypto’96
24. Daniel J. Bernstein. “Cache-timing attacks on AES”. 2005
25. Paul Kocher, Joshua Jaffe and Benjamin Jun. ”Differential Power Analysis”. Crypto’99
26. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. CHES’01
27. Daniel Genkin, Adi Shamir and Eran Tromer. “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”. CRYPTO’14
28. Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic and Jean-Pierre Seifert. “Simple Photonic Emission Analysis of AES Photonic Side Channel Analysis for the Rest of Us”. CHES’12
29. David Brumleya and Dan Boneh, “Remote timing attacks are practical”. Computer Networks’05.
30. Preneel, Bart and Paar, Christof and Pelzl, Jan. “Understanding cryptography: a textbook for students and practitioners”. Springer 2009

References [4]

31. Bellare M, Rogaway P. Optimal asymmetric encryption EUROCRYPT'94
32. https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding
33. https://en.wikipedia.org/wiki/Montgomery_modular_multiplication
34. https://en.wikipedia.org/wiki/Karatsuba_algorithm
35. https://github.com/stoutbeard/crypto
36. Osvik D A, Shamir A, Tromer E. Cache attacks and countermeasures: the case of AES[M]//Topics in Cryptology–CT-RSA 2006. Springer Berlin Heidelberg, 2006: 1-20.
37. Chongxi Bao and Ankur Srivastava. "3D Integration: New Opportunities in Defense Against Cache-timing Side-channel Attacks". ICCD'15.
38. Gorka Irazoqui, Thomas Eisenbarth and Berk Sunar, "S$A: A Shared Cache Attack that Works Across Cores and Defies VM Sandboxing—and its Application to AES", Oakland'15
39. Yarom Y, Falkner K. Flush+ reload: a high resolution, low noise, L3 cache side-channel attack. USENIX Security 14
40. Gruss D, Maurice C, Wagner K. Flush+ Flush: A Stealthier Last-Level Cache Attack. arXiv preprint arXiv:2015

References [5]

41. Roberto Guanciale, Hamed Nemati, Christoph Baumann, and Mads Dam, “Cache Storage Channels: Alias-Driven Attacks and Verified Countermeasures”, S&P’16
42. Zhenghong Wang and Ruby B. Lee “New cache designs for thwarting software cache-based side channel attacks”, ISCA 2007.
43. Kong J, Aciicmez O, Seifert J P, et al. Deconstructing new cache designs for thwarting software cache-based side channel attacks, ACM workshop on Computer security architectures. 2008
44. Costan V, Lebedev I, Devadas S. Sanctum: Minimal Hardware Extensions for Strong Software Isolation[J].
45. Danfeng Zhang, Yao Wang, G. Edward Suh and Andrew C. Myers. “A Hardware Design Language for Timing-Sensitive Information-Flow Security”. ASPLOS’15
46. Mangard S, Oswald E, Popp T. Power analysis attacks: Revealing the secrets of smart cards[M]. Springer Science & Business Media, 2008.
47. Costan V, Devadas S. Intel SGX Explained[J]. IACR Cryptology ePrint Archive, 2016, 2016: 86.
48. Hoheisel A. Side-Channel Analysis Resistant Implementation of AES on Automotive Processors[D]. Master’s thesis, Ruhr-University Bochum, Germany (June 2009), 2009.
49. Tiri K, Verbauwhede I. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation[C]//Proceedings of the conference on Design, automation and test in Europe-Volume 1. IEEE Computer Society, 2004: 10246.
50. Trichina E. Combinational Logic Design for AES SubByte Transformation on Masked Data[J]. IACR Cryptology ePrint Archive, 2003, 2003: 236.

References [6]

51. John T M, Haider S K, Omar H, et al. Connecting the Dots: Privacy Leakage via Write-Access Patterns to the Main Memory[J]. arXiv preprint arXiv:1702.03965, 2017.
52. Mangard S, Pramstaller N, Oswald E. Successfully attacking masked AES hardware implementations[C]//International Workshop on Cryptographic Hardware and Embedded Systems. Springer Berlin Heidelberg, 2005: 157-171.
53. Nikova S, Rechberger C, Rijmen V. Threshold implementations against side-channel attacks and glitches[C]//International Conference on Information and Communications Security. Springer Berlin Heidelberg, 2006: 529-545.
54. Ventzi Nikov, presentation on "Threshold Implementation"18.03.2016
55. Kutzner S, Nguyen P H, Poschmann A. Enabling 3-share threshold implementations for all 4-bit s-boxes[C]//International Conference on Information Security and Cryptology. Springer International Publishing, 2013: 91-108.
56. Gross H, Mangard S, Korak T. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order[C]//Cryptographers' Track at the RSA Conference. Springer, Cham, 2017: 95-112.
57. Gras B, Razavi K, Bosman E, et al. ASLR on the Line: Practical Cache Attacks on the MMU[J]. NDSS (Feb. 2017), 2017.