Can randomized mapping secure instruction caches from side-channel attacks?

Fangfei Liu, Hao Wu and Ruby B. Lee
Princeton University

June 14, 2015

Outline

- Motivation and Background
  - Data cache attacks and defenses
  - Random-Fixed mapping and Newcache
- Constructing instruction cache attacks
- Effectiveness of Random-Fixed mapping to I-cache attacks
- Power and system performance
- Conclusions

Cache side channel attacks

- Information leakage by exploiting timing difference between cache hit and cache miss
- The cache is a **shared resource**
  - Cache state affects, and is affected by all processes
  - Cache contention
- Cryptanalysis through **cache address leakage**
  - No disclosure of data stored in the cache
- The "**metadata**" leaks information about memory access patterns
  - Which addresses are being accessed
  - Memory access patterns depend on the secret key

Data cache attacks

- Secret dependent memory indexing
  - e.g., AES table lookup

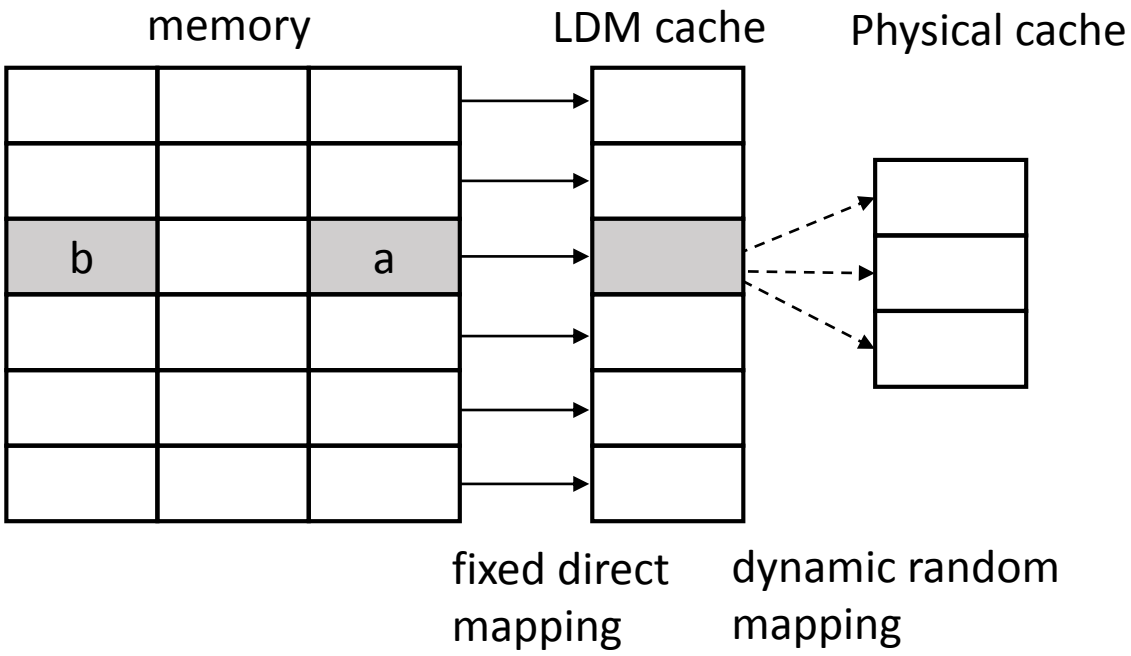
```
s0 = GETU32(in     ) ^ rk(0);
s1 = GETU32(in + 4) ^ rk(1);
s2 = GETU32(in + 8) ^ rk(2);
s3 = GETU32(in + 12) ^ rk(3);
t0 = Te0[s0>>24] ^ Te1[(s1>>16)&0xff] ^ Te2[(s2>>8)&0xff] ^ Te3[s3&0xff] ^ rk(4)
```

Defenses for data cache

- Secure cache design
  - Target contention based attacks
  - Partitioning the cache
    - Eliminate cache contention
    - Static partitioning or dynamic partitioning
    - Performance degradation due to cache underutilization
  - Randomizing the memory-to-cache mapping
    - Allow cache contention
    - No information can be extracted
    - Negligible performance degradation

Newcache

- Random-Fixed mapping



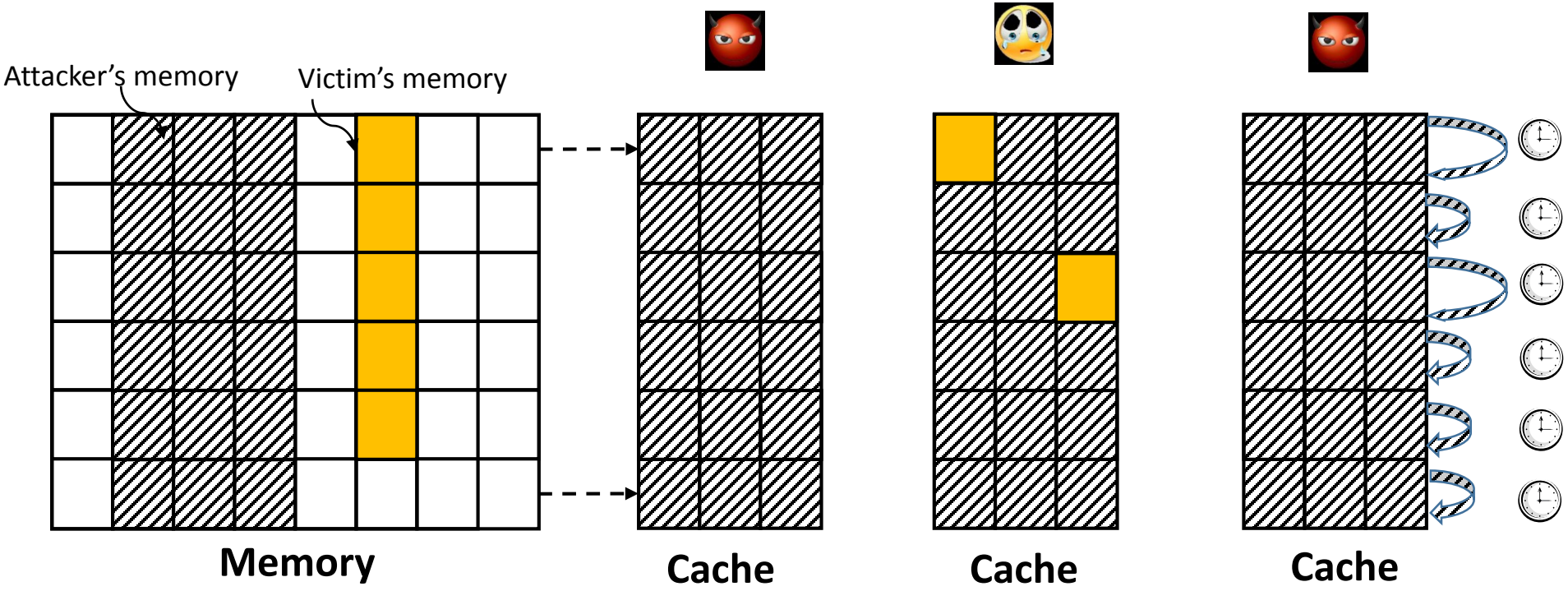
Can Newcache-like randomized mapping defeat contention based attacks against I-cache?

Instruction cache attacks

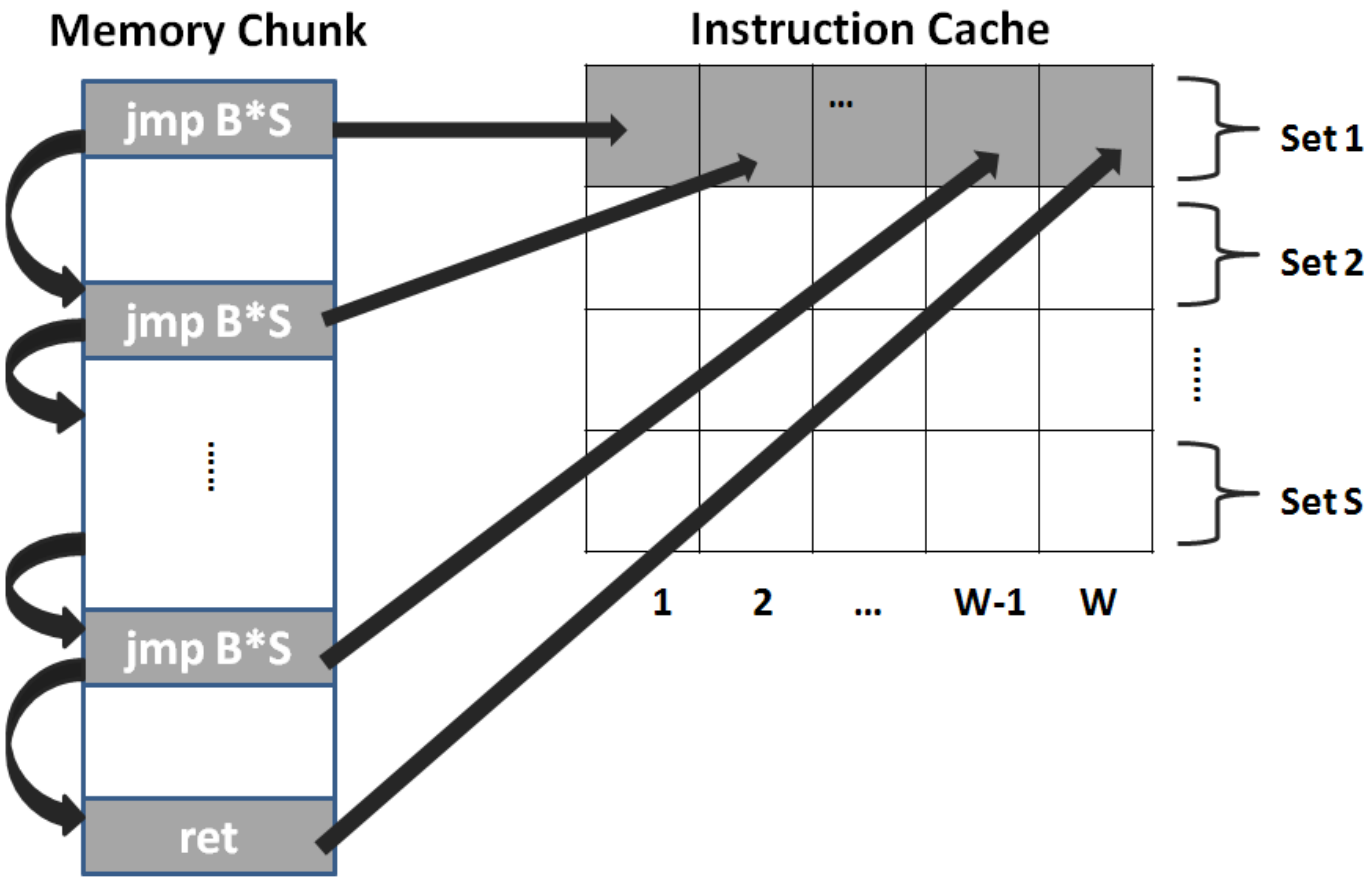
- Secret-dependent instruction paths
  - Public-key cryptography
  - Applications processing secret information, e.g., password, credit card information

```
if (secret == 1) {
        code block 1;
 } else {
        code block 2;
 }
```

Prime-Probe Attacks



Construct Prime-Probe attacks on I-cache



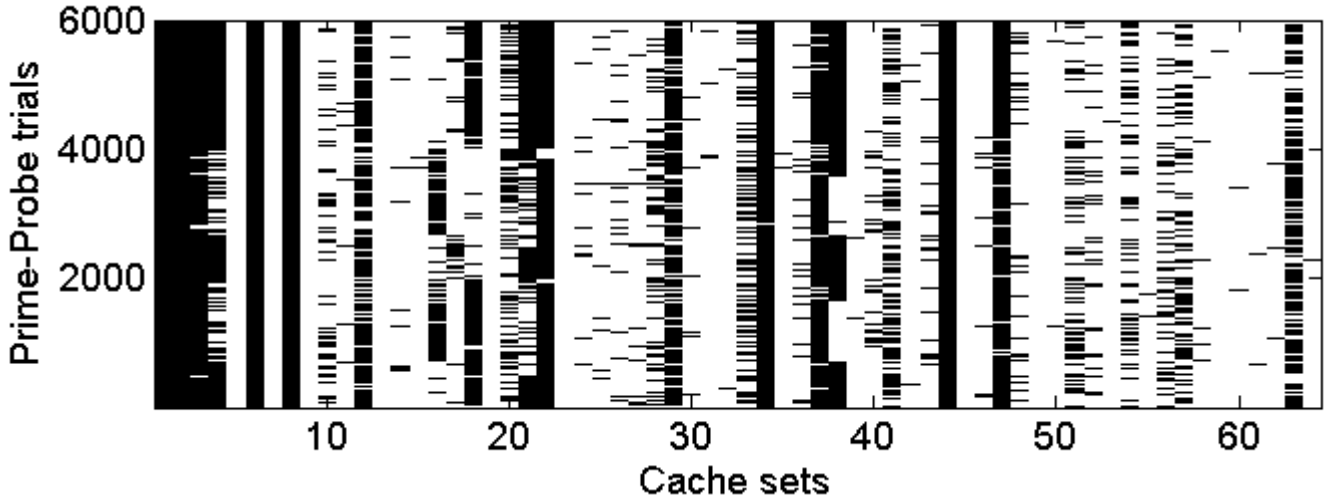
PRINCETON UNIVERSITY

Prime-Probe attacks on modular exponentiation

- Implementation of modular exponentiation
  - Basic "square and multiply" algorithm
  - Exponent bits scanned from MSB to LSB (left to right)

```
Let k = bitsize of d

Let s = m

For i = k−2 down to 0

    Let s = s*s (SQUARE)
    Let s = s mod n (REDUCE)

    If (bit i of d) is 1 then
        Let s = s*m (MULTIPLY)
        Let s = s mod n (REDUCE)
    End if

End for
```

Cache footprint and SVM classification matrix



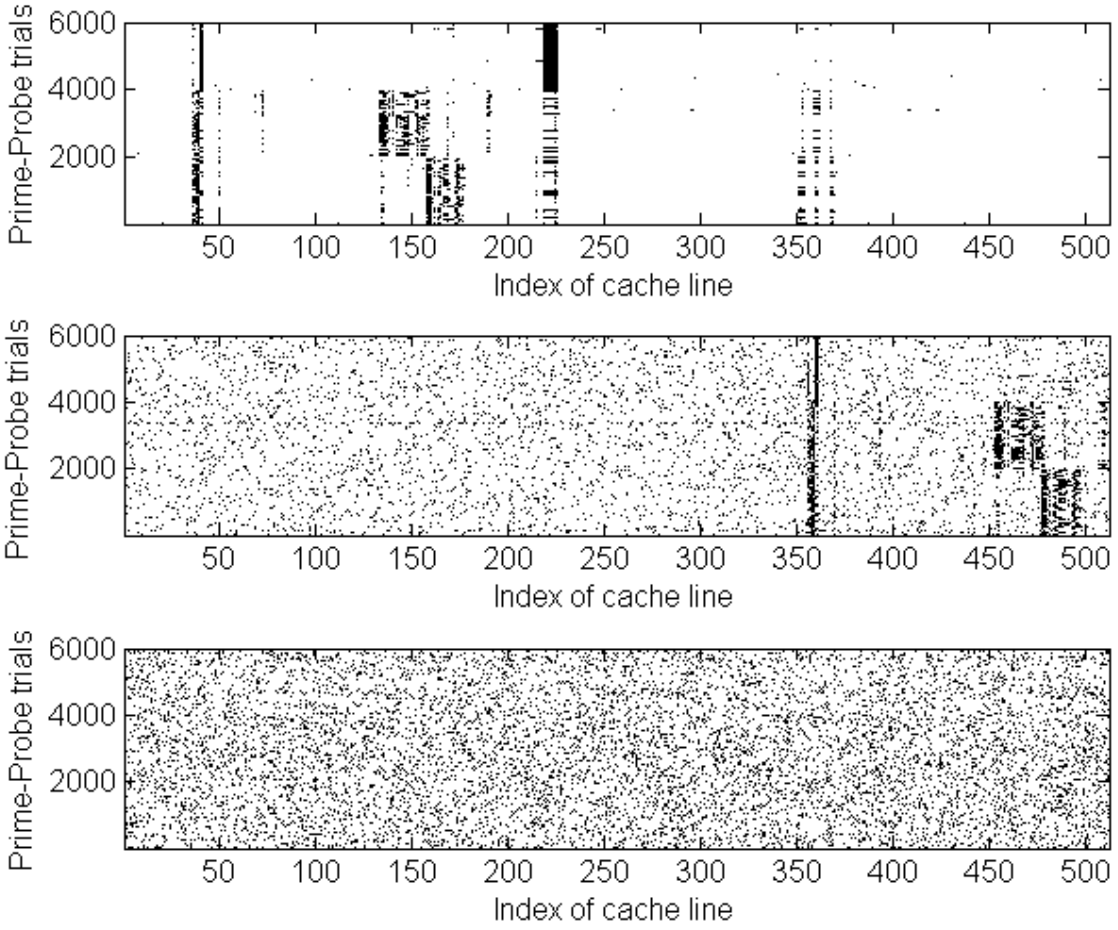
	Classification			Classification Accuracy
	Square	Multiply	Reduce	
Op: Square	**3470 (0.87**)	189 (0.05)	332 (0.08)	**90.2%**
Op: Multiply	375 (0.09)	**3587 (0.90)**	38 (0.01)	
Op: Reduce	92 (0.02)	148 (0.04)	**3760 (0.94)**	

Effectiveness of randomized mapping on I-cache

- Characterizing methodology
  - Perform Prime-Probe attacks on gem5 simulator
  - To achieve fine-grained preemption and Prime-Probe operations
    - Hack simulator to execute dummy memory accesses for the probe operations at some fixed time interval
  - Use SVM classification matrix as a metric
  - Accuracy -> 33%, cannot distinguish S, M, R operations

Cache foot print with Random-Fixed mapping

- Varying size of logical direct mapped (LDM) cache
  - 1X Cache size
  - 4X Cache size
  - 16X Cache size



PRINCETON UNIVERSITY

SVM classification matrix for Random-Fixed mapping

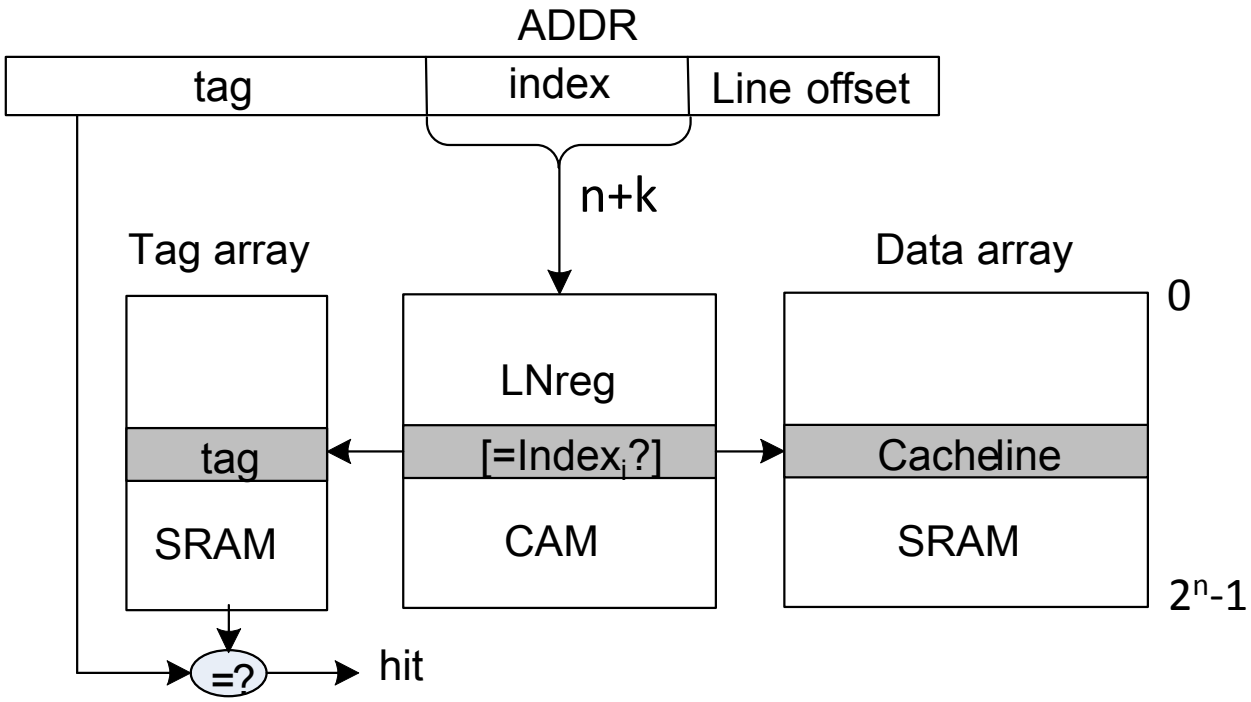
		Square	Multiply	Reduce	accuracy
1X cache size	Square	**3978 (0.99)**	0 (0.00)	22 (0.01)	**98.7%**
	Multiply	1 (0.00)	**3983 (1.00)**	17 (0.00)	
	Reduce	10 (0.00)	107 (0.03)	**3883 (0.97)**	
4X cache size	Square	**3840 (0.96)**	5 (0.00)	155 (0.04)	**96.4%**
	Multiply	5 (0.00)	**3850 (0.96)**	145 (0.04)	
	Reduce	62 (0.02)	61 (0.02)	**3877 (0.97)**	
16X cache size	Square	**1143 (0.29)**	940 (0.24)	1917 (0.47)	**41.0%**
	Multiply	1153 (0.29)	**1098 (0.27)**	1748 (0.44)	
	Reduce	696 (0.17)	620 (0.16)	**2684 (0.67)**	

Summary of results

- Random-Fixed mapping
  - Provides a different spectrum of cache designs ranging from direct mapped cache to fully associative cache
    - LDM size = 1X cache size: direct mapped cache
      - all fixed mapping
      - Still vulnerable to Prime-Probe attacks
    - LDM size = memory size: fully associative cache
      - all randomized mapping
      - Completely defeats Prime-Probe attacks
  - Increasing LDM cache size reduces the probability of the index conflicts, and hence increases the difficulty of attacks

Holistic hardware implementation

- Replace address decoder of conventional cache with CAM



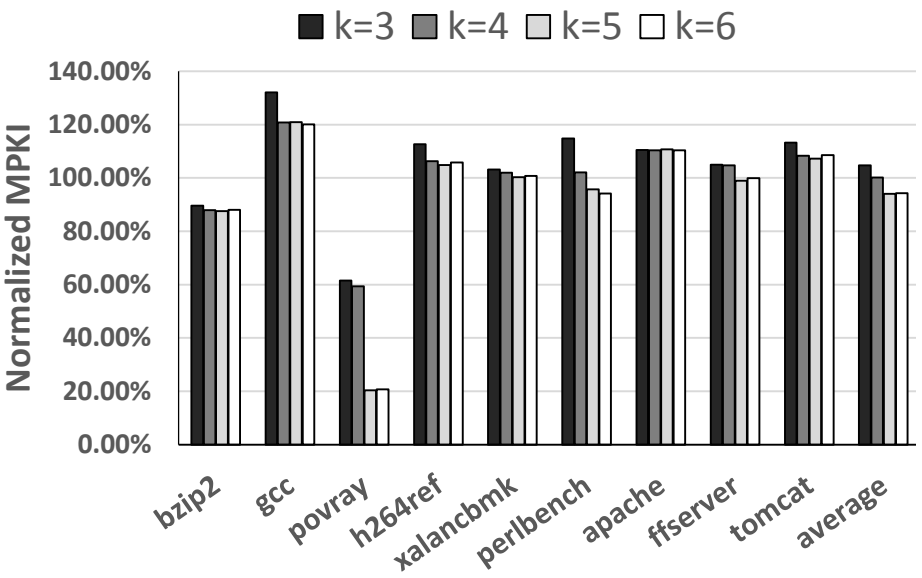
k = extra index bits

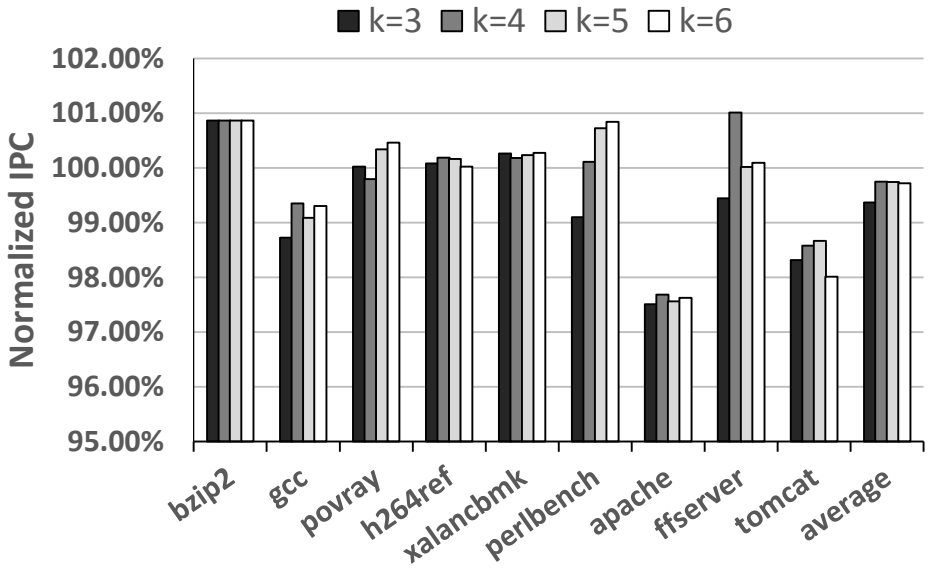
PRINCETON UNIVERSITY

Power and latency

- Hierarchical NAND-CAM
  - Consume much less power than NOR-CAM
- Implemented the whole cache using 65nm CMOS process
  - Slightly faster than the 8-way set associative cache
  - Slightly less power

System performance





- Workloads are chosen to have large working sets (stress instruction cache)
- Overall performance degradation is less than 0.3%

PRINCETON UNIVERSITY

Conclusions

- Ideal randomized mapping can completely defeat contention based attacks
- Random-Fixed mapping provides full design spectrum from direct-mapped cache to fully associative cache
- Increasing the size of the LDM (or the number of extra index bits *k)* increases the difficulty of attacks
- Newcache replacement for both the I-cache and D-cache in processors will prevent cache side channel attacks without degrading either system performance or cache physical performance

Thank You!

Q&A

PRINCETON UNIVERSITY