

---

# NEWCACHE: SECURE CACHE ARCHITECTURE THWARTING CACHE SIDE-CHANNEL ATTACKS

---

NEWCACHE IS A SECURE CACHE THAT THWARTS CACHE SIDE-CHANNEL ATTACKS, PREVENTING THE LEAKAGE OF CRITICAL INFORMATION. THE AUTHORS PRESENT AN IMPROVED DESIGN OF NEWCACHE, IN TERMS OF SECURITY, CIRCUIT DESIGN, AND SIMPLICITY. THEY SHOW NEWCACHE'S SECURITY AGAINST A SUITE OF CACHE SIDE-CHANNEL ATTACKS AND DESIGN A TEST CHIP TO PROVE ITS FEASIBILITY. NEWCACHE'S SYSTEM PERFORMANCE IS AS GOOD AS CONVENTIONAL SET-ASSOCIATIVE CACHES.

..... An increasing amount of sensitive data and proprietary programs are now stored in cyberspace. With the escalating number of cyberattacks, it is crucial that we protect the confidentiality and integrity of data and programs in our networked computers. Although strong cryptography can be used to encrypt and authenticate data, this protection is rendered useless if the secret crypto keys can be leaked out. It turns out that this can be done rather easily through cache side-channel attacks, which are software attacks on hardware caches. Today, all processors with caches—from embedded systems to smartphones to cloud computers—are susceptible to these cache side-channel attacks.

Software memory isolation mechanisms, like virtual machine or process isolation, cannot prevent cache side-channel attacks because the underlying hardware caches are still shared. Also, it is important to understand that leakage of critical information

through cache side-channel attacks happens with correctly functioning caches. Unlike software security vulnerabilities that are due to software bugs, cache side-channel attacks are not due to hardware flaws. They use the caches' intrinsic behaviors, wherein cache hits are fast and cache misses are slow.

Although researchers have proposed software solutions to cache side-channel attacks, they incur significant performance degradation—reported as  $3\times$  to  $10\times$  slowdown.<sup>1</sup> Also, their security is not assured, because software cannot directly control the hardware-managed caches, which can change with different implementations of the processor-cache subsystem. Furthermore, although it might be possible to change the software for well-known crypto libraries, it is not possible to change all legacy software with embedded crypto routines, nor other software using secret or sensitive data or code. It may be more feasible to provide security for legacy programs by changing the

**Fangfei Liu**  
**Hao Wu**  
Princeton University

**Kenneth Mai**  
Carnegie Mellon University

**Ruby B. Lee**  
Princeton University

underlying cache architecture at the hardware level.

Hardware solutions proposed to thwart cache side-channel attacks have typically used some form of cache partitioning with reduced overall system performance.<sup>2–4</sup> In contrast, Newcache<sup>5</sup> is a novel secure cache architecture based on randomized mappings, implementing a “moving target defense” strategy for securing hardware caches. Although Newcache has a random replacement algorithm, its design and testing is fully deterministic. It has the promise of defeating cache side-channel attacks without degrading performance.<sup>5</sup> However, when considering technology transfer to the industry, it is necessary to perform additional security testing, extensive performance evaluation for today’s computing environments (such as cloud computing and smartphones), and design a test chip to prove the feasibility of a Newcache-style cache. Hence, we thoroughly evaluated Newcache’s system-level security and performance using a detailed simulation of Newcache on a full system simulator. We also designed a test chip to prove the feasibility of Newcache and to measure its physical characteristics, such as access latency, power, and area, compared to a conventional set-associative (SA) cache of the same size. We report the results in this article.

## Background

We give a brief introduction to cache side-channel attacks and the Newcache architecture.

### Overview of Cache Side-Channel Attacks

Because caches are shared resources, cache states affect and are affected by all processes. Therefore, one process can infer the cache usage of another process through cache contention. In conventional SA caches with fixed memory-to-cache mapping and least-recently-used replacement, all cache contentions are deterministic, which enables an attacker process to deduce the memory addresses accessed by a victim process—and, therefore, to deduce the secret information if the memory addresses are secret-dependent.

Cache side-channel attacks on both level-1 (L1) caches<sup>6–8</sup> and last-level caches<sup>9</sup> have been shown. In this article, we discuss the attacks against the L1 caches, including both the data

cache (D-cache) and the instruction cache (I-cache), because L1 caches are closest to the processor, so attacks against them are the fastest and most dangerous. We assume the attacker can share the L1 cache with the victim through simultaneous multithreading (SMT) or preemptive scheduling.

### Dynamic Randomized Mapping and Newcache

A promising approach to defeat cache side-channel attacks is to randomize the memory-to-cache mapping at runtime so that an attacker cannot extract useful information from observing cache contention. Randomized mapping can be achieved using a fully associative cache with a random replacement algorithm. However, the fully associative cache is slow and power hungry. Newcache provides a practical way to achieve randomized mapping with much lower impact on latency and power.<sup>5,10–12</sup> Conceptually, this is done by a level of indirection. The memory address is first mapped to a logical direct mapped (LDM) cache, and each LDM cache line is then mapped in a fully associative and randomized way to a physical cache line. Although this two-step mapping is done conceptually, physically, the LDM cache does not actually exist, and the mapping is done by accessing a content addressable memory (CAM; see Figure 1). Each CAM entry is called a line number register (LNreg), which stores the logical cache line number in the LDM cache of the associated physical cache line. In essence, the LNregs replace the static address decoder in a direct-mapped cache architecture. Although all other aspects of a cache’s architecture have been optimized for performance or power reasons over the decades, a key novelty of the Newcache architecture is that it is the first to consider such replacement of a cache’s address decoder.

Given the existence of LNregs in Newcache, it is easy to increase the width of the LNregs by a few bits, called  $k$  extra index bits. This corresponds to an LDM cache that is  $2^k$  times larger than the physical cache, which can improve both its performance and its security.

A cache access compares the index bits within the desired memory address with the contents of all the LNregs for a match. On a match with the contents of LNreg <sub>$i$</sub> , called an

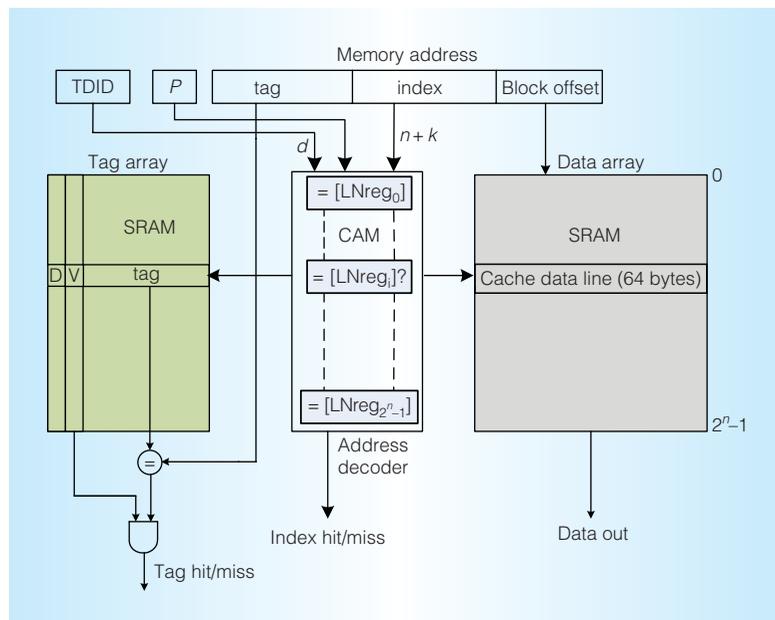


Figure 1. Block diagram of Newcache. The fixed address decoder of a direct-mapped cache is replaced by a dynamic (inverse) line-number mapper, which can be implemented by content addressable memory (CAM).

*index hit*, Newcache checks that the associated cache tag matches the rest of the memory address, called a *tag hit*, and simultaneously reads out the associated cache line. If the tag part does not match (*tag miss*), the associated cache line is replaced with the new cache line, as for a direct-mapped cache. On an index miss, a random cache line is selected for replacement, giving a dynamic, randomized memory-to-cache mapping.

To eliminate potential fixed-cache conflicts during a tag miss between a victim process and an attacker process, Newcache also provides each trust domain with a disjoint mapping (identified by a Trust Domain Identifier [TDID]) so that no conflicts will ever occur between two processes from different trust domains. Newcache can also eliminate the cache conflicts between protected and unprotected memory regions within a process, by concatenating the TDID with a “P”

bit to indicate a protected cache line. We improved the original Newcache design<sup>5</sup> by moving the P bit from the tag array to the LNregs. This deflects a subtle exploit<sup>10</sup> and simplifies the replacement algorithm.<sup>5</sup> Note that the CAM implementation of Newcache (which we’ll describe in the “Test Chip Design” section) also differs from the circuit design proposed for the original Newcache.<sup>5</sup>

### Security Testing

Newcache targets all the L1 cache attacks that exploit cache contention. We identified different classes of cache side-channel attacks: access-based versus timing-based attacks, and attacks on instructions versus attacks on data (see Table 1). We developed an attack suite of cache side-channel attacks, including known attacks on vulnerable cryptographic algorithms (see Table 2), to evaluate Newcache’s security. Our security testing results show that Newcache defeats all known cache side-channel attacks on the L1 D-cache and I-cache due to cache contention.

Prime-Probe and Evict-Time are two common attack techniques.<sup>6</sup> Prime-Probe is an access-driven attack in which the attacker measures how the victim’s memory accesses impact its own memory accesses. Evict-Time is a timing-driven attack in which the attacker measures the total execution time of the victim’s encryption operation.

A Prime-Probe attack works as follows:

- Prime: An attacker *A* fills the cache (or just one or more security-critical cache sets) with its own data.
- Idle: *A* waits for a Prime-Probe interval while the victim process *V* gains control of the processor and uses the cache.
- Probe: *A* gains control of the processor again and measures the time to access the same cache sets to learn *V*’s cache activity.

Cache type being attacked	Access-based attack	Timing-based attack
Data cache	Prime-Probe	Evict-Time
Instruction cache	Prime-Probe	N/A

**Table 2. Summary of attack suite and attack results on Newcache.**

Cache type	Attack	Results	
		Set-associative (SA) cache	Newcache
Data cache	Prime-Probe attack against first-round Advanced Encryption Standard (AES)	Attack succeeds (0.4209*)	Attack is defeated (0.0032*)
	Evict-Time attack against first-round AES	Attack succeeds (0.8068*)	Attack is defeated (0.0217*)
Instruction cache	Prime-Probe attack against modular exponentiation	Attack succeeds (90.2% <sup>†</sup> )	Attack is defeated (48.1% <sup>†</sup> )

\*Pearson correlation coefficient; <sup>†</sup>accuracy of classification.

If  $V$  uses some cache sets during the Prime-Probe interval, some of  $A$ 's cache lines in these cache sets will be evicted, which will cause a longer load time for those cache sets during  $A$ 's probe phase. The Prime-Probe technique can attack both the D-cache and the I-cache.

An Evict-Time attack works as follows:

- **Evict:** An attacker  $A$  fills one specific cache set with its own data.
- **Time:**  $A$  triggers the victim process to perform the security-critical operation (for example, encrypting one block of plaintext) and measures  $V$ 's execution time.

If the victim accesses the evicted cache set, its execution time tends to be statistically higher than when it does not access the evicted cache set, due to the victim having a cache miss.

We modeled Newcache in gem5,<sup>13</sup> a cycle-accurate simulator, as a cache object that can be used for any cache in the cache hierarchy. Our attack suite can be run on gem5 on top of the system with Newcache or conventional caches.

### Prime-Probe Attack on L1 D-Cache

We use the Prime-Probe attack against the first-round encryption of the Advanced Encryption Standard 128 (AES-128). The attack code primes the D-cache by reading data from an array before it calls the AES encryption library call to encrypt one block of random plaintext. Then the attack code probes the array and measures the time to access each cache set (for the SA cache) or each cache line

(for Newcache). For the SA cache, the array can be the same size as the D-cache. For Newcache, we use a smaller array with the same size as the AES tables to avoid self-eviction during Prime and Probe operations.

Figure 2 shows the D-cache heat maps for the SA cache and Newcache. Each point represents the average probe time over 220 trials, with the lighter color representing a longer probe time. Because we use an all-zero key (for simplicity without losing generality), we can see a clear bright straight line if the attack succeeds, which is the case for the eight-way SA cache (see Figure 2a). However, there is no bright straight line in the heat map for Newcache, which indicates that Newcache can defeat the attack (see Figure 2b).

Although the visual results are clear, we also confirm them with quantitative results. We can calculate the Pearson correlation coefficient between the attacker's measurements (the D-cache heat map) and ground truth. The ground truth is the expected D-cache pattern—that is, a piecewise straight line with each segment containing 16 points (a 64-byte cache line contains 16 AES table entries). The closer the coefficient is to zero, the less accurately the attacker is observing patterns. The correlation coefficients in parentheses for the D-cache in Table 2 agree with the visual heat map.

### Evict-Time Attack on L1 D-Cache

The Evict-Time attack is also performed against the first-round encryption of AES-128. The attack code first evicts one cache line containing AES table entries out of the D-cache, then invokes a library call to

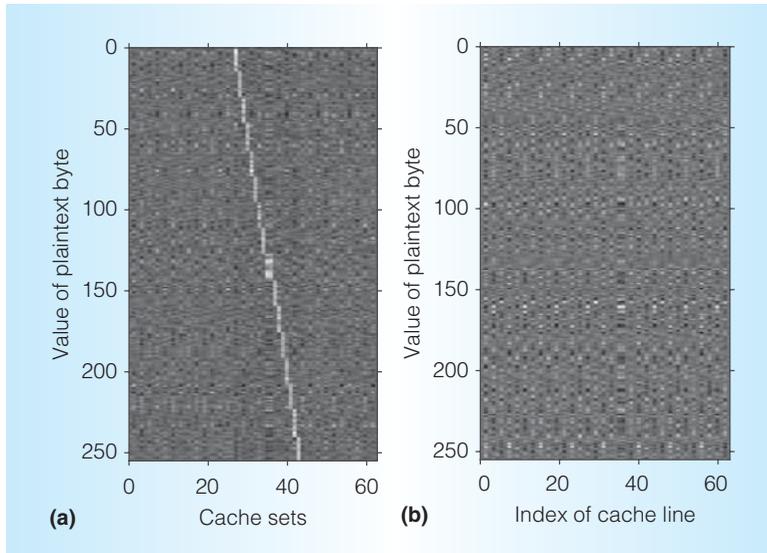


Figure 2. Data cache heat map (average probe times) for (a) eight-way set-associative (SA) cache and (b) Newcache. The bright straight line indicates that the attack succeeded.

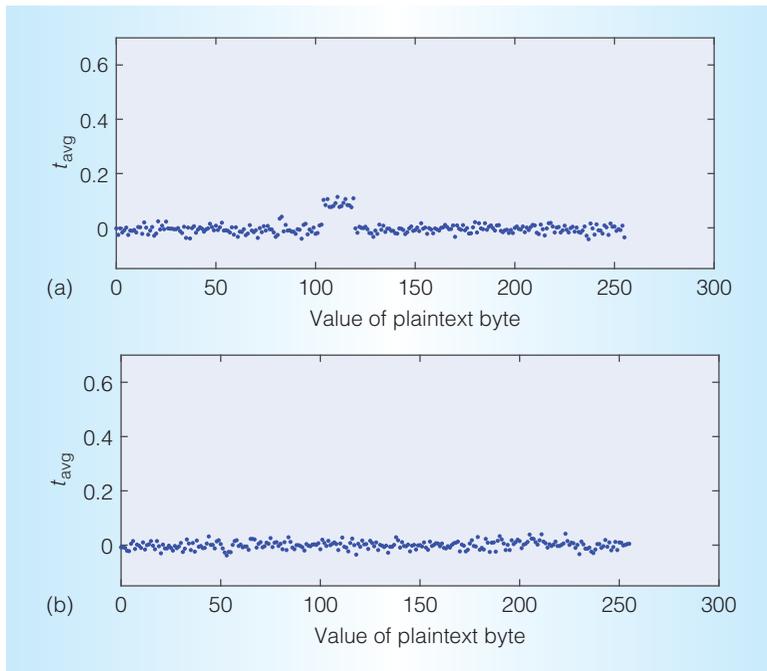


Figure 3. Evict-Time attack results for (a) eight-way SA cache and (b) Newcache. A significantly higher average encryption time for certain plaintext byte values indicates that the attack succeeded.

encrypt one AES block with random plaintext and measures the encryption time. If the attack can succeed, the average encryption time over 220 trials will be higher for certain

plaintext values (due to cache misses), thus revealing certain key bits. This is true for the eight-way SA cache (see Figure 3a), but there is not a significantly higher average time for Newcache (see Figure 3b).

We can also quantify a design's vulnerability to Evict-Time attacks by correlating the attacker's measurements with the expected ground truth values ("1" for points with significantly higher average encryption time, "0" for others), as shown in Table 2.

### Prime-Probe Attack on L1 I-Cache

Performing Prime-Probe attacks on the I-cache is trickier than on the D-cache. To prime or probe one cache set for a  $W$ -way SA cache, the attacker must execute  $W \text{ jmp } S$  instructions, which jump to the next  $\text{jmp } S$  instructions, except for the last block, which has a return instruction. ( $S$  is the size, in bytes, of one way of the cache.) The attacker jumps to the first  $\text{jmp } S$  instruction from the main program, and the  $W$  memory blocks containing  $\text{jmp } S$  instructions will be fetched into the I-cache set, one by one, before returning to the main program.

We use a Prime-Probe attack against the modular exponentiation algorithm in libcrypto v1.5.3. Modular exponentiation is the main computation in many public-key algorithms, such as RSA and ElGamal. It can be implemented as a square-and-multiply algorithm that scans the exponent bits from left to right. For each bit, the algorithm performs a square operation and a modular reduce operation. If the exponent bit is 1, it performs an extra multiply operation and modular reduce operation. The sequence of operations executed by the algorithm can leak information about the exponent, which is the private key used in public-key ciphers.

For each Prime-Probe trial, the attacker gets a cache footprint, which is a vector of timings, one timing per cache set (or cache line, for Newcache). The footprint can be classified as one of the square, multiply, or reduce operations using a multilabel support vector machine (SVM) classifier. Figure 4 shows the classification matrices (also called confusion matrices) for the attack, for the set-associative cache, and for Newcache. The diagonal entries of each  $3 \times 3$  matrix are the correct classifications. The classification accuracy is 90.2

	Square	Multiply	Reduce		Square	Multiply	Reduce
<b>Square</b>	3,479 (87%)	189 (5%)	332 (8%)	<b>Square</b>	1,570 (39%)	1,545 (39%)	885 (22%)
<b>Multiply</b>	375 (9%)	3,587 (90%)	38 (1%)	<b>Multiply</b>	1,484 (37%)	1,869 (47%)	647 (16%)
<b>Reduce</b>	92 (2%)	148 (4%)	3,760 (94%)	<b>Reduce</b>	799 (20%)	874 (21%)	2,327 (58%)
Accuracy = 90.2%				Accuracy = 48.1%			
<b>(a)</b>				<b>(b)</b>			

Figure 4. SVM classification matrix for (a) SA cache and (b) Newcache. The matrix entries are the number of times SVM classified a pattern as the operation in the column label when it was actually the operation in the row label. The diagonal entries are correct classifications. The number in parenthesis represents the percentage (probability) of that classification, with the three entries in a row summing to 100%. The accuracy of SVM classification is also listed.

percent for the SA cache, which indicates that the three operations can be easily distinguished. In contrast, the classification accuracy for Newcache is only 48.1 percent, which means it would be extremely difficult for further offline analysis to extract key bits correctly.<sup>7</sup>

In addition to contention-based cache side-channel attacks, there are also reuse-based attacks,<sup>14</sup> such as cache collision attacks.<sup>8</sup> Although Newcache cannot defeat these attacks, it is about one order of magnitude harder to attack than the conventional SA cache. We can easily enhance Newcache to defeat reuse-based attacks by modifying the D-cache's cache controller to use a random fill fetch policy, as proposed by Fangfei Liu and Ruby Lee.<sup>14</sup>

## System Performance

We completed a thorough performance evaluation on smartphone and cloud computing benchmarks using the gem5 simulator in full-system mode. For the smartphone benchmarks, we used four benchmark suites: Bbench, OxBench, CoreMark, and Mibench. For cloud computing, we used six of the most common cloud computing workloads: web server (apache), database server (mysql), mail server (bhm), file server (smbd), streaming server (ffserver), and application server (tomcat). We ran the server benchmarks in gem5 dual-system mode, simulating both the client and server.

We found that smartphone benchmarks show no performance impact when using Newcache versus the SA cache, so we show

results for cloud server benchmarks only and compare them with the SPEC2006 benchmarks typically used in computer architecture conferences. Table 3 gives the baseline simulator configuration, which is the same as for our security evaluations discussed earlier.

### Newcache Used as D-Cache

We compare the performance of the baseline configuration in Table 3 with one where the D-cache is replaced with Newcache. Figures 5a and 5b show the D-cache miss rate and overall performance, in instructions per cycle (IPC), normalized to the baseline eight-way SA cache, respectively, for different numbers of extra index bits  $k$  in Newcache.

*Server benchmarks.* The random replacement of Newcache (for example, web server or database server) could increase the D-cache miss rate, because some frequently used data could be randomly evicted. However, Newcache could also decrease the D-cache miss rate for other benchmarks, such as the file server and application server, because it changes the conflicting sets of memory lines that map to the same cache line. On average, the miss rate decreases slightly with an increasing  $k$ . Newcache's impact on the IPC is negligible. On average, the overall performance is degraded by less than 1 percent for different values of  $k$  (from 3 to 6).

*SPEC benchmarks.* The SPEC benchmarks have larger variations for the D-cache miss rate than the server benchmarks, and some of the benchmarks (such as mcf, astar, and

**Table 3. Baseline simulator configurations**

Parameter	Value
L1 data cache	8-way SA, 32 Kbytes, 4 cycles
L1 instruction cache	4-way SA, 32 Kbytes, 4 cycles
L2 cache	8-way SA, 256 Kbytes, 10 cycles
L3 cache (last-level cache)	16-way SA, 2 Mbytes, 35 cycles
Cache line size	64 bytes
Memory size, latency	2 Gbytes, 200 cycles

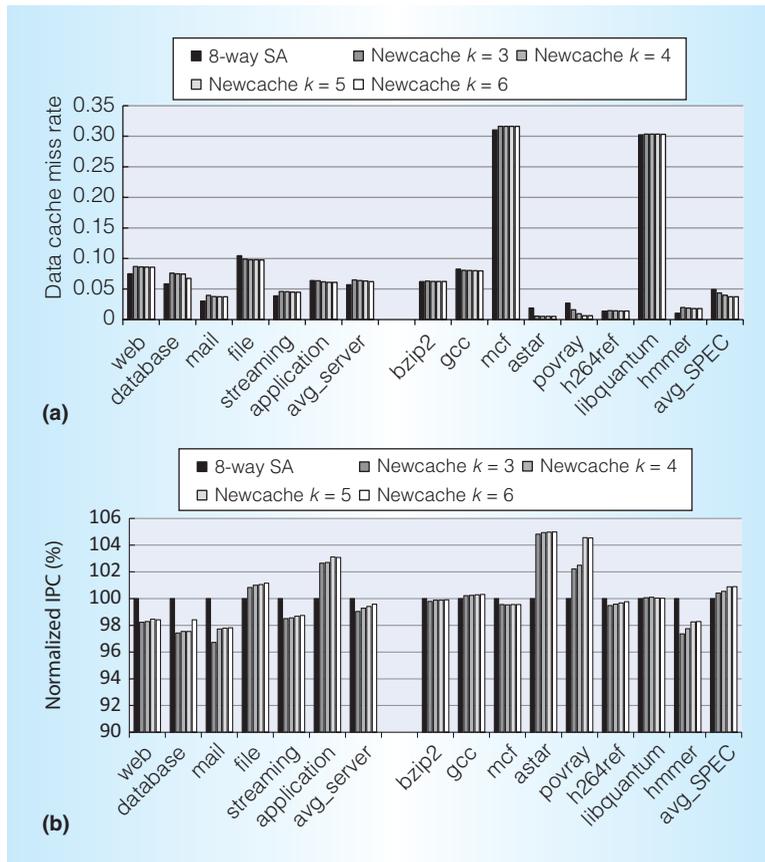


Figure 5. Performance of Newcache as the data cache. (a) D-cache miss rate (lower is better). (b) Normalized instructions per cycle (higher is better).

libquantum) are memory-intensive. Newcache can significantly reduce the miss rate for some SPEC benchmarks, such as *astar* and *povray*, which could improve the IPC by about 5 percent. The largest performance degradation is within 3 percent for *hammer*. On average, the overall performance actually improves slightly, by 0.5 percent, for different values of  $k$  (from 3 to 6).

### Newcache Used as I-Cache

We next replaced only the I-cache with Newcache (see Table 3). Figures 6a and 6b show the I-cache miss rate and IPC, normalized to the baseline four-way SA I-cache, respectively.

*Server benchmarks.* Server benchmarks have relatively high I-cache miss rates, due to their large instruction working sets. On average, Newcache induces about a 3 percent higher I-cache miss rate than the four-way SA cache. Increasing  $k$  decreases the average miss rate slightly. We notice that for some workloads (for example, mail server, streaming server, and application server), Newcache as the I-cache has no penalty. The impact to the overall performance in terms of IPC is very small. For  $k > 3$ , no workloads cause IPC degradation larger than 2 percent.

*SPEC benchmarks.* Generally, SPEC benchmarks have a much lower I-cache miss rate (except for *gcc*), so the impact on overall performance is small. The IPC variations are all within 0.5 percent, except for *povray*, which can significantly reduce the already-low I-cache miss rate and improve the IPC by 3 percent when  $k > 3$ .

### Recommendations

To summarize, Newcache with  $k = 4$  extra index bits, for both the D-cache and the I-cache, can achieve performance equivalent to conventional SA caches of the same size.

For efficient cache coherency management, we suggest using a directory-based cache coherence protocol, or inclusive caches, where snooping at the L1 cache is not required. These kinds of cache subsystems are also the most common today.

### Test Chip Design

We designed a test chip to establish the feasibility of Newcache's physical design using a 65-nm CMOS process. The test chip has a 32-Kbyte Newcache and a 32-Kbyte eight-way SA cache on the same chip for direct comparison. Figure 7 shows the die microphotograph. We used circuit-level optimizations, such as a hierarchical NAND-type CAM, which detects a match instead of a mismatch

and consumes much less power than a conventional NOR-type CAM. The eight-way SA cache accesses the tag and data array serially, so eight tags are read out in parallel but only one word (64 bits) of data is read out from the data array to conserve power. There is a small area overhead of 10 percent for the cache (which implies a negligible overall processor core overhead of less than 1 percent). The power overhead was about 20 percent for the test chip we measured, where the SA cache was optimized for lower power, as described earlier. If the SA cache was optimized for higher performance instead, where the eight tags and eight data are read out in parallel, the SA cache would incur higher power, and Newcache would have a lower power overhead. The security and performance benefits of Newcache also suggest that new designs for lower-power CAMs should be further explored for caches. Importantly, Newcache's access latency was limited by the data array common to both caches but not by the CAM.

We adopted a CAM-based design for the dynamic memory-to-cache mapping for Newcache when used as L1 caches, because of the sensitivity to access latency for L1 caches for performance reasons. For L2 and larger caches, which have longer access times, a CAM-based implementation is not necessary, and future work can explore other implementations of a Newcache-like architecture for dynamic mapping.

Our security evaluation shows that Newcache can completely defeat all known contention-based attacks for both the D-cache and the I-cache, and our test chip design shows that Newcache is feasible and comparable to an eight-way SA cache of the same size. Furthermore, our extensive performance evaluation shows that Newcache performs as well as conventional set-associative caches for contemporary computing environments such as cloud computing, desktops, and smartphones—sometimes even slightly better. Hence, our recommendation to processor vendors is to replace current SA L1 I-caches and D-caches with secure caches like Newcache. This will prevent critical information leakage through cache side-channel attacks without impacting performance, even for legacy software.

MICRO

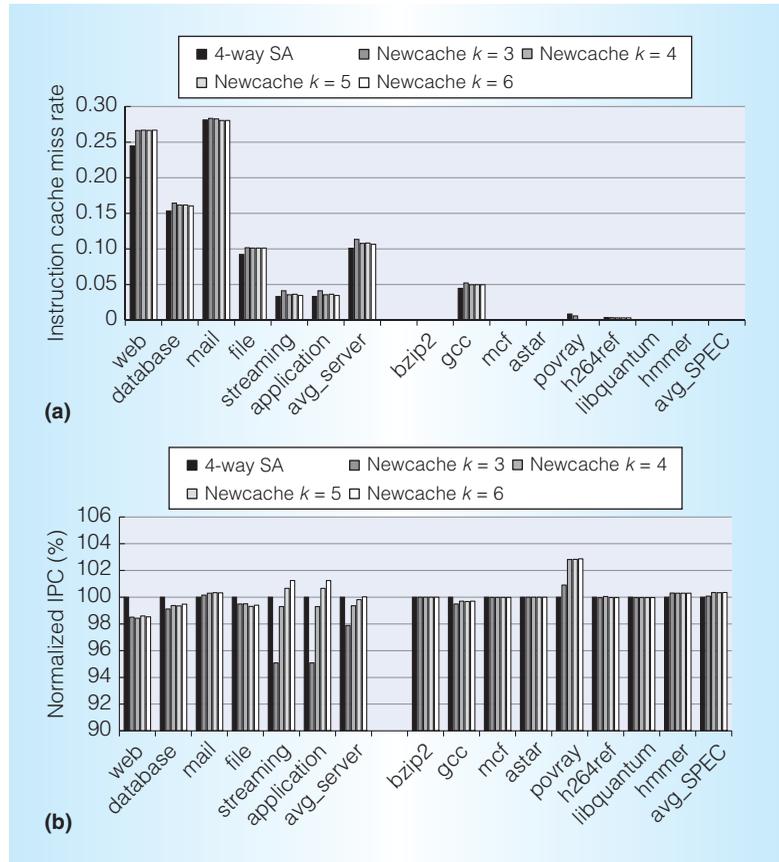


Figure 6. Performance of Newcache as the instruction cache. (a) I-cache miss rate (lower is better). (b) Normalized instructions per cycle (higher is better).

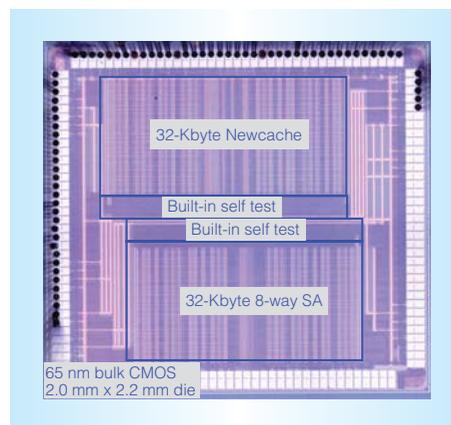


Figure 7. Die microphotograph of Newcache test chip. The majority of area and power is dominated by the SRAMs, which is common to both Newcache and the SA cache, each of 32 Kbytes with its own built-in self-test circuitry.

## Acknowledgments

This work was supported by Dept. of Homeland Security/Air Force Research Laboratory grant FA8750-12-2-0295 and the attack metrics by National Science Foundation STARSS award 1526493.

## References

1. E. Brickell et al., *Software Mitigations to Hedge AES against Cache-Based Software Side Channel Vulnerabilities*, report 2006/52, IACR Cryptology, 2006.
2. Z. Wang and R.B. Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks," *Proc. 34th Ann. Int'l Symp. Computer Architecture*, 2007, pp. 494–505.
3. L. Domnitser et al., "Non-monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks," *ACM Trans. Architecture and Code Optimization*, vol. 8, no. 4, 2012, article 35.
4. D. Page, *Partitioned Cache Architecture as a Side-Channel Defence Mechanism*, report 2005/280, IACR, 2005.
5. Z. Wang and R.B. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture*, 2008, pp. 83–93.
6. D.A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," *Proc. Cryptographers' Track at RSA Conf. Topics in Cryptology*, 2006, pp. 1–20.
7. Y. Zhang et al., "Cross-VM Side Channels and Their Use to Extract Private Keys," *Proc. ACM Conf. Computer and Communications Security*, 2012, pp. 305–316.
8. J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks against AES," *Proc. 8th Int'l Conf. Cryptographic Hardware and Embedded Systems*, 2006, pp. 201–215.
9. F. Liu et al., "Last-Level Cache Side-Channel Attacks Are Practical," *Proc. IEEE Symp. Security and Privacy*, 2015, pp. 605–622.
10. F. Liu and R.B. Lee, "Security Testing of a Secure Cache Design," *Proc. 2nd Workshop Hardware and Architectural Support for Security and Privacy*, 2013, article 3.
11. F. Liu, H. Wu, and R.B. Lee, "Can Randomized Mapping Secure Instruction Caches from Side-Channel Attacks?" *Proc. 4th Workshop Hardware and Architectural Support for Security and Privacy*, 2015, article 4.
12. B. Erbagci et al., "A 32kB Secure Cache Memory with Dynamic Replacement Mapping in 65nm Bulk CMOS," *Proc. IEEE Asian Solid-State Circuits Conf.*, 2015; doi:10.1109/ASSCC.2015.7387501.
13. N. Binkert et al., "The gem5 Simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, 2011, pp. 1–7.
14. F. Liu and R.B. Lee, "Random Fill Cache Architecture," *Proc. 47th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2014, pp. 203–215.

**Fangfei Liu** is a PhD candidate in the Department of Electrical Engineering at Princeton University. She received an MS in electrical engineering from Shanghai Jiao Tong University. Contact her at [fangfei@princeton.edu](mailto:fangfei@princeton.edu).

**Hao Wu** is a CPU performance validation architect at Soft Machines. He received an MS in electrical engineering from Princeton University, where he completed the work for this article. Contact him at [haow.princeton@gmail.com](mailto:haow.princeton@gmail.com).

**Kenneth Mai** is a principal systems scientist in the Electrical and Computer Engineering Department at Carnegie Mellon University. He received a PhD in electrical engineering from Stanford University. He is a member of IEEE. Contact him at [kenmai@andrew.cmu.edu](mailto:kenmai@andrew.cmu.edu).

**Ruby B. Lee** is the Forest G. Hamrick Professor in the Department of Electrical Engineering at Princeton University. She received a PhD in electrical engineering from Stanford University. She is a Fellow of IEEE and ACM and is on the advisory board of *IEEE Micro*. Contact her at [rblee@princeton.edu](mailto:rblee@princeton.edu).

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.