RISC-V Secure Caches: Demo on FPGA

caslab.csl.yale.edu

Shuwen Deng, Wenjie Xiong, and Jakub Szefer

Computer Architecture and Security Laboratory, Yale University, USA

1. Project Overview and Contributions

During past years, researchers have shown a plethora of timing-based side channels, especially in processor caches. All of these attacks have demonstrated that it is possible to extract sensitive information via the timing-based side channels, and often the focus is on extracting cryptographic keys.

To address the threat of these cache timing side-channel attacks, researchers have presented a number of secure processor cache designs. However, there is no unified hardware platform to measure their performance and no comprehensive method to illustrate their security. Our RISC-V based FPGA implementation of secure caches is the first step towards a hardware platform that could be used to evaluate different types of caches and even test full-system security.

- We present a demo showing RISC-V secure caches on the FPGA board, which aim to mitigate timing-based cache side-channel attacks. This is a microarchitecture level secure cache framework design based on RISC-V Rocket Chip generator using Chisel hardware construction language.
- We present Partition Locked cache (PL cache) [1], which we realized in FPGA hardware to show its security and performance.

2. Rocket Chip

Rocket Chip [2] is an open-source SoC design generator. It can generate synthesizable RTL from Chisel hardware construction language. It is composed of a library of sophisticated generators, including the ones for cores, caches, and interconnects into an integrated SoC. An example of the instance is shown in Figure 1. The 5-stage in-order RISC-V core generator supports page-based virtual memory, data/instruction caches, and a front-end with branch prediction. It can be configured for the different board (e.g. ZC706) and generate Verilog code and corresponding booting binary used for the board.

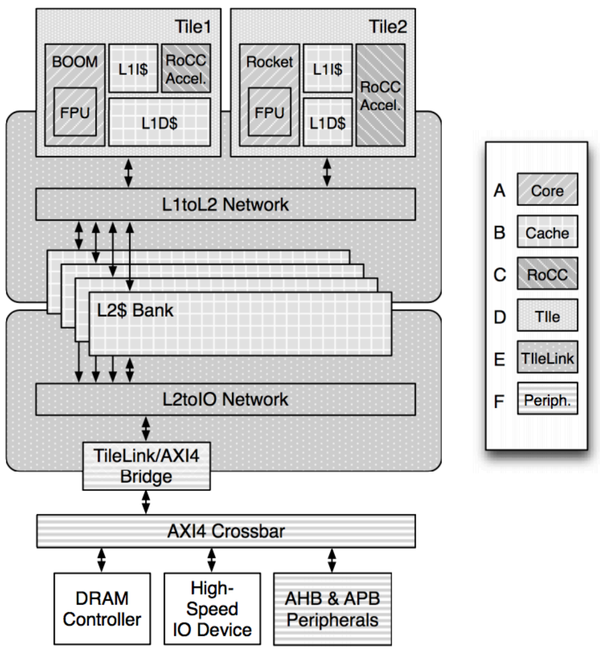

Fig 1. An example of a Rocket Chip [2] instance.

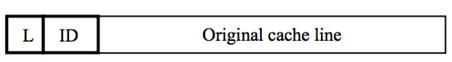

Fig 2. A cache line of the PL cache.

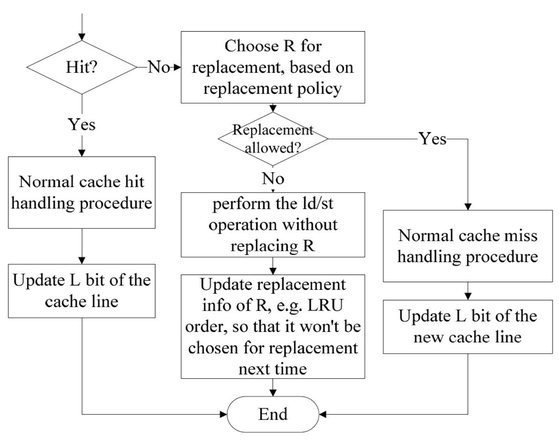

Fig 3. Access handling procedure for the PL cache.

3. Secure Partition Locked (PL) Cache

PL cache [1] provides isolation by partitioning cache based on cache blocks. It extends each cache block with a process ID and a lock status bit (L) (Figure 2). The ID and L bits are controlled by the extended load and store instructions which allow the programmer and compiler to set or reset the lock bit through the use of the right load or store instruction. The cache replacement policy for the PL cache is shown in Figure 3.

4. Timing Side-Channel Attacks

The attacker (in the same or separate CPU as victim's) can themselves access the cache by making memory accesses, or drive the victim to access the cache by making memory accesses, e.g. request victim to do some known computation or both. The attacker usually knows what code the victim is executing, e.g. type of encryption algorithm, but does not know the victim's specific secrets. The attacker aims to extract some secret information. Figure 4 shows the example of Flush+Reload [3] attack.

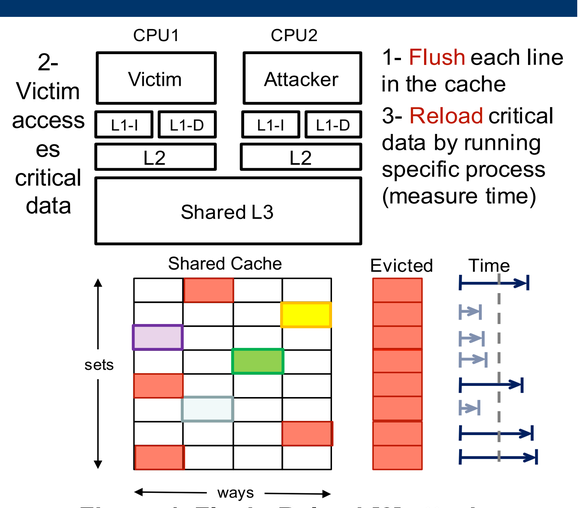

Figure 4. Flush+Reload [3] attack.

5. Security Evaluation

Evaluation Setup

The hardware setup for the demonstration includes Xilinx Zynq-7000 SoC ZC706 FPGA board, a Linux CPU (host computer) connected to a display, power supply for FPGA, a USB JTAG cable, and a USB UART cable.

5. Security Evaluation (cont'd)

We performed the Flush+Reload cache timing side-channel attack on AES RISC-V Rocket Chip on FPGA, with both the case of using normal cache and PL cache (The correct way to use PL cache is to preload and lock the AES table first and unlock the AES table at the end of the program).

AES 128 encryption requires a 16-byte input n using a 16-byte key k. In "AddRoundKey", each byte of n is combined with a block of k using bitwise XOR. In "SubBytes", each byte is replaced with another according to a lookup table.

Targeted set: Set 34, 15th byte of the k ($k[14]$).	
When k is all 0 (Figure 5(a)):	When k is none-0 (Figure 5(b)):
Set 34 plaintext range: 3584~3599=(256*14+0)~(256*14+15) $n0[14]$: 0~15	Set 34 plaintext range: 3824~3839 =(256*14+240)~(256*14+255) $n1[14]$: 240~255
$k0[14]$: 0×00 (known) Time of AES Table Te's look-up: $\mathrm{t}(\mathrm{Te}[n0[14]\ \mathrm{XOR}\ k0[14]]) = \mathrm{t}(\mathrm{Te}[0])$ short. (higher 4 bit result of XOR)	$k1[14]$ (unknown) Time of AES Table Te's look-up: $\mathrm{t}(\mathrm{Te}[n1[14]\ \mathrm{XOR}\ k1[14]])$ short. Thus, $k1[14] =$ $(n0[14]\ \mathrm{XOR}\ k0[14])\mathrm{XOR}\ n1[14] = 0\times ff$

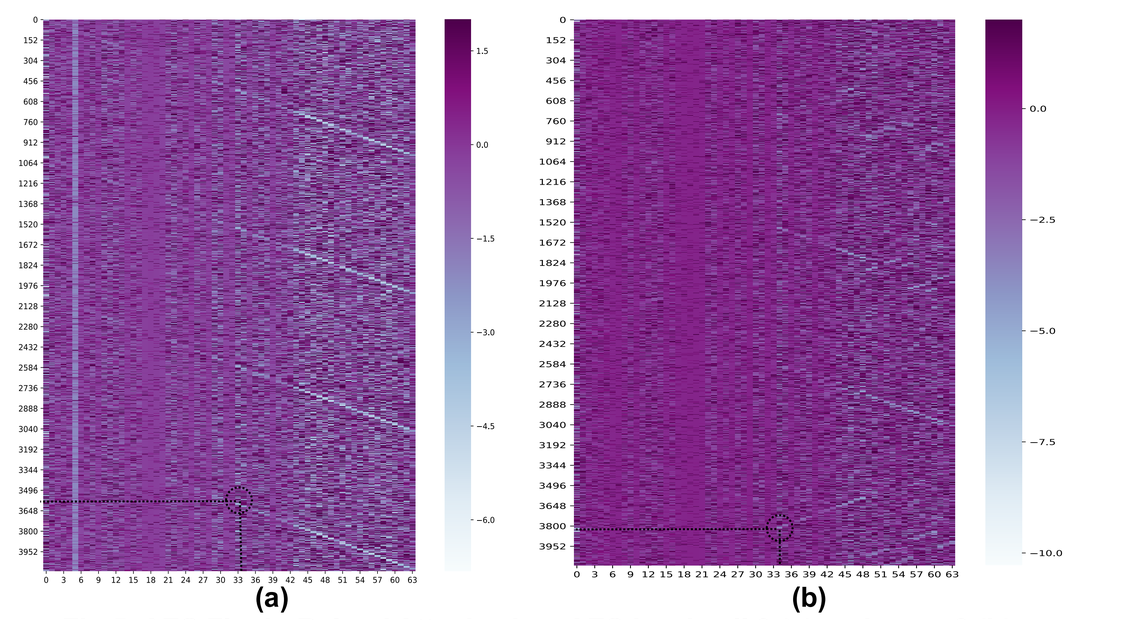

Fig 5. AES Flush+Reload Attack when AES key is all 0 (a) and none-0 (b).

Figure 6 shows that normal cache cannot prevent AES attacks like Flush+Reload, while PL cache is able to hide the pattern related to the key value and protect it.

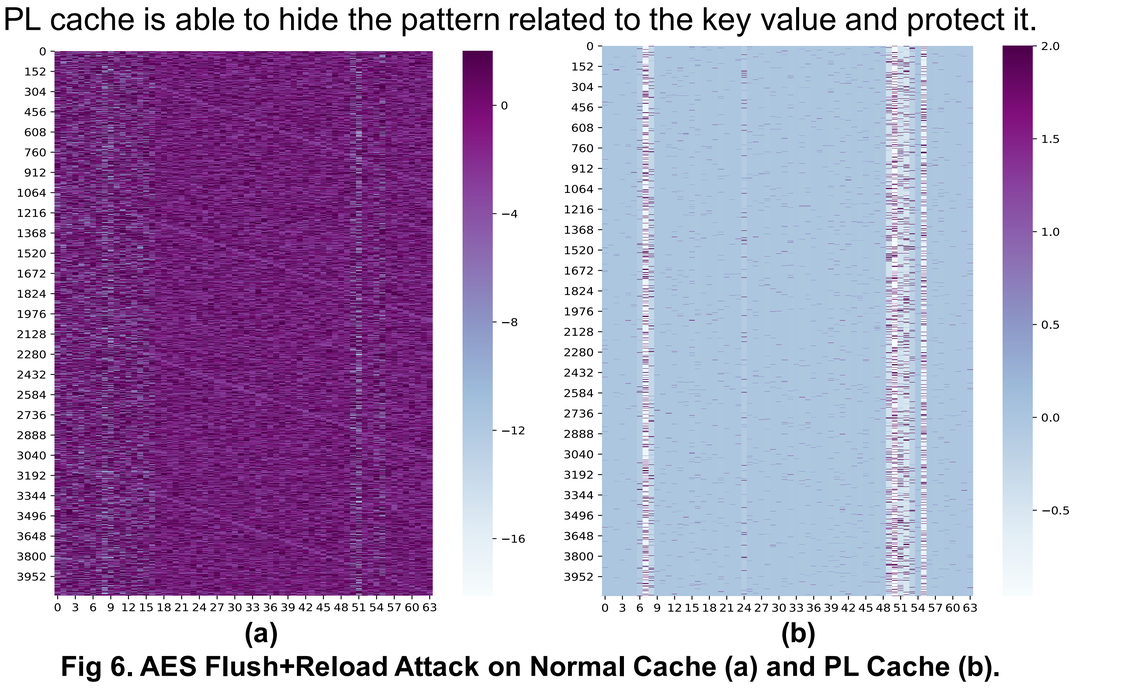

Fig 6. AES Flush+Reload Attack on Normal Cache (a) and PL Cache (b).

6. PL Cache vs. Normal Cache

	Metric	32-set, 4-way	64-set, 2-way	fully associative (128 way)
Normal Cache	Slice LUTs	36611	36400	53457
	Slice Registers	20201	20179	24161
	Block RAM	22	15	6
	DSP Usage	15	15	15
	Total On-Chip Power (W)	2.065	2.060	2.084
PL Cache	Metric	32-set, 4-way	64-set, 2-way	fully associative (128 way)
	Slice LUTs	37056	36620	53967
	Slice Registers	20382	20351	25991
	Block RAM	26	17	6
	DSP Usage	15	15	15
	Total On-Chip Power (W)	2.082	2.059	2.098

[1]. Wang, Z., & Lee, R. B. (2007). New cache designs for thwarting software cache-based side channel attacks. ACM SIGARCH Computer Architecture News, 35(2), 494-505.

[2] Asanovic, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., ... & Karandikar, S. (2016). The rocket chip generator. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17.

[3] Yarom, Y., & Falkner, K. (2014). FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In 23rd {USENIX} Security Symposium ({USENIX} Security 14) (pp. 719-732).