CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017

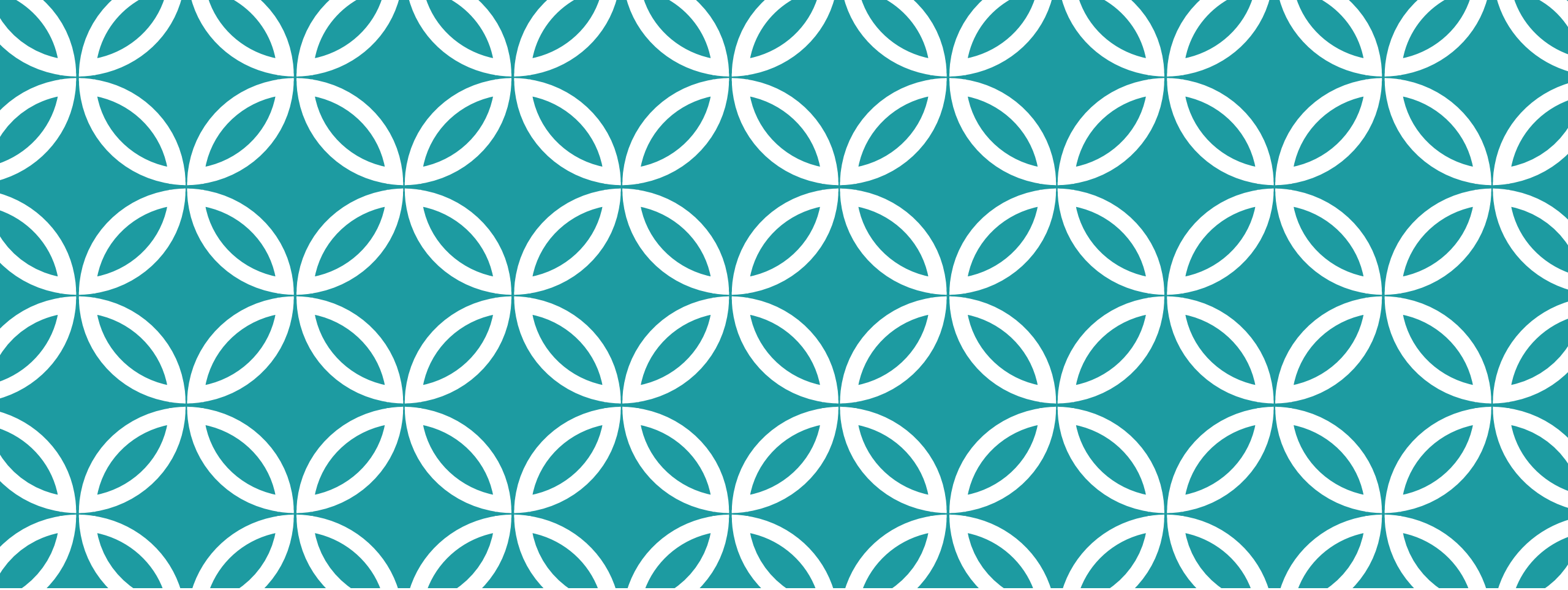
Assignment 4 (developed by Chenglu & Kamran)

Cache Side Channel Attack

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

Department of Electrical & Computer Engineering
University of Connecticut

UCONN



Getting Started

SimpleSim Simulator Setup

- It is assumed that:
  - You have Virtual Box installed on your system.
  - You have already setup the "ECE5451-VM" provided in previous assignment.
  - You are able to run test application by running "**`make counters_bench_test`**"
  - You are familiar with SimpleSim directory structure explained in previous assignment.
- Please revisit Assignment 2 slides if you have problems with above mentioned steps.

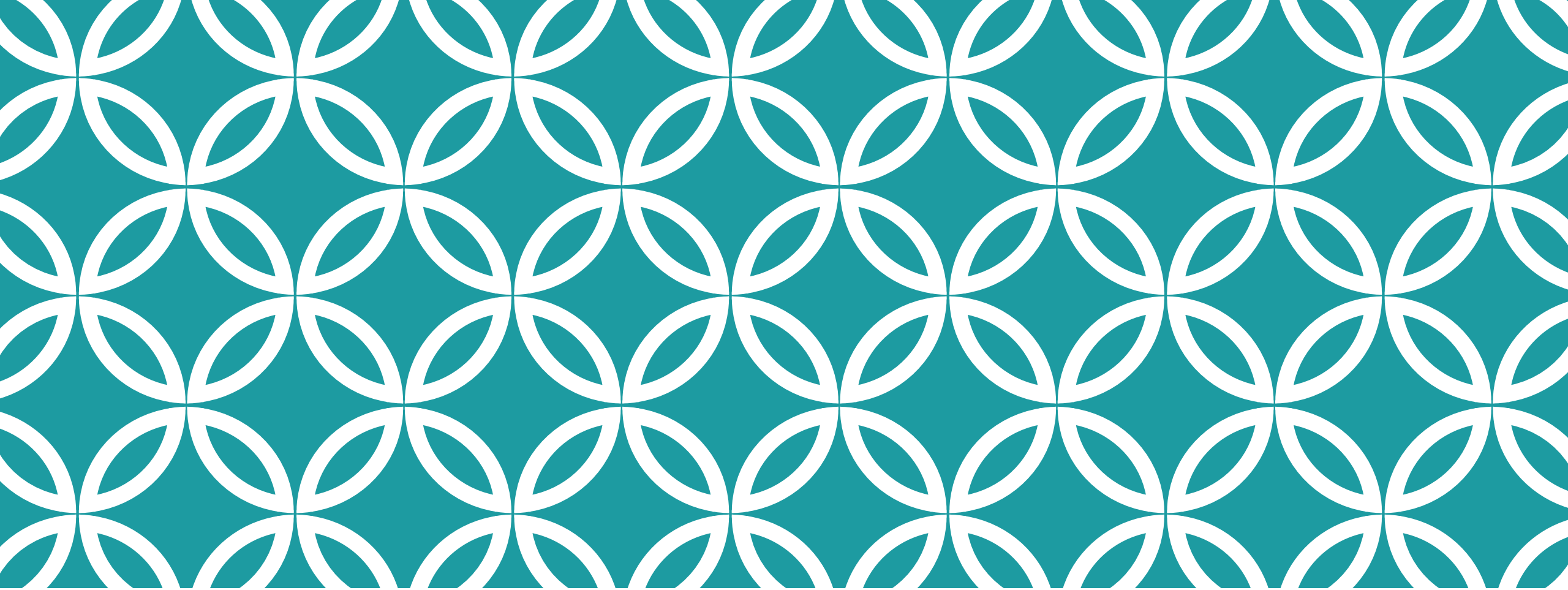
Getting the latest source files

- Login the ECE5451-VM using the following credentials:
  - Username: student
  - Password: student5451
- Enter the SimpleSim_Public directry:
  - `[student@ECE5451-VM ~]$ cd SimpleSim_Public/`
    `[student@ECE5451-VM ~/SimpleSim_Public]$`
- Run “git pull”. This will download all the latest source files in your local machine.
  - `[student@ECE5451-VM ~/SimpleSim_Public]$ git pull`
- Several updates and new files will be added under `SimpleSim_Public/simulator` and `SimpleSim_Public/tests/benchmarks/countersMT` directories.
- Run “make clean” to remove old executables, and then run “make” to compile the simulator
  - `[student@ECE5451-VM ~/SimpleSim_Public]$ make clean`
  - `[student@ECE5451-VM ~/SimpleSim_Public]$ make`

Running a Test Application

- To run a test application, do
  - `make countersMT_bench_test`
- You'll notice the following:
  - Two threads, attacker and victim, are alternatively executing, hence sharing the same cache/DRAM resources.

```
thread begin 0. _total_threads(1)
Starting Application...
thread begin 1. _total_threads(2)
Thread(1) suspending itself. Time(143720)
Victim Task Started.
Thread(0) suspending itself. Time(155990)
Thread(1) Resuming. Time(155990)
Attacker Task Started.
Thread(1) suspending itself. Time(193880)
Thread(0) Resuming. Time(193880)
...
...
Victim: Time(577530)
Thread(0) suspending itself. Time(586230)
Thread(1) Resuming. Time(586230)
Attacker: Time(589835)
thread end 1 code 0, _total_threads(1)
Thread(0) Resuming. Time(599775)
Done...!
thread end 0 code 0, _total_threads(0)
```



Lab Task: Prime + Probe Attack

Prime + Probe Attack

- 1. Attacker fills one selected cache set
- 2. Attacker triggers the execution of victim process
- 3. Victim accesses the cache
- 4. Attacker accesses the same cache set he/she has filled and measures the access time to figure out if it is a cache hit or miss

Task

- You need to implement your attacker and victim threads in tests/benchmarks/countersMT/countersMT.cc
  - Notice: in this lab you are implement the application which will run on top of the simulated computer. You don't need to modify the simulator at all.
- Two threads have been created for you in the file. You just need to modify two functions corresponding to both the threads:
  - attacker_task()
  - victim_task()
- You need to construct some data structures (e.g. an array), one in each of the two functions, such that you can find two different virtual addresses being mapped into the same cache set. Although these two threads are in one process, they are not allowed to access the same virtual address to create cache interference.
- To guarantee the order of memory accesses (attacker, victim, attacker), you can use a lock to communicate between these two threads. But you should be aware of the context switching.
- To get the time information from the simulator, use SimGetTime().

Address Translation

- Notice that the cache in this simulator is a physically indexed cache, which means that the cache set number is calculated based on its physical address. But in the application, you only have control of its virtual address. So it is very critical for you to understand address translation to successfully launch this attack.
- We have a page table that is initially "empty".
- When a virtual address comes in, its virtual page number is computed (by simply discarding 12 LSBs of the virtual address, i.e. page size is 4kB).
- Since page table is initially empty, this means this virtual page is not yet mapped to any physical page. At this point, this virtual page is mapped to the first (available) physical page (i.e. physical page starting at 0x0000). The offset within the page does not change (i.e. 12 LSBs of the virtual address remain the same).
- Next, if another address translation request comes in from the same virtual page, since this page already has a valid entry in the page table, it will be simply assigned the corresponding physical page.
- If another page is requested which currently does not have a valid entry in the page table, the next available physical page (i.e. physical page starting at 0x1000) is assigned to it, and so on.
- So essentially, the physical page allocation is at First Come First Serve basis.
- For more details about address translation, please find it in VirtualToPhysicalAddress function in simulator_main.cc

Tips

- In order to make sure that the page mapping is created as you want, we suggest you to initialize all the page mapping in the main function before two threads starting execution.
  - Notice that the program itself may need to access some memory which is not a part of your constructed data structure. They will create some other page mappings, so please be aware of this.
- Also, you can print out the virtual address and physical address from the simulator to double check the cache set number you are targeting.

Demo

- You need to perform two prime+probe attacks. One of them shows the scenario that the victim does access this primed cache set. The other one shows the scenario that the victim does not access this primed cache set.
- Sample Result:
- For scenario 1: probe time is 50005, cache miss
- For scenario 2: probe time is 5, cache hit