Composable Cachelets: Protecting Enclaves from Cache Side-Channel Attacks

Daniel Townley, *Peraton Labs;* Kerem Arıkan, Yu David Liu, and Dmitry Ponomarev, *Binghamton University;* Oğuz Ergin, *TOBB University of Economics and Technology*

https://www.usenix.org/conference/usenixsecurity22/presentation/townley

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

This artifact includes the gem5 simulator and McPAT files that are used for performance impact and area/power estimation for the paper "Composable Cachelets: Protecting Enclaves from Cache Side-Channel Attacks". The aforementioned frameworks will generate result files which we explain how to extract the relevant results in the appendix. In order to provide the artifact with minimal dependencies, we have provided the reviewers with a safe remote access to a server that contains all of the relevant dependencies, benchmarks, executables, source code and scripts, all of which can be inspected, overwritten and executed.

A.2 Artifact check-list (meta-information)

All of the following material in the checklist is included in the server. There is nothing to be downloaded. We are going to be mentioning them for context.

- **Algorithm:** The main algorithms we introduce are the allocation and the remapping code in the gem5 simulator. The bulk of our cachelet allocation algorithm is contained at `gem5/src/mem/cache/tags/indexing_policies`. In this directory, we have a base class in `base.cc` where the allocation and class constructor reside. For the remapping algorithm, we modify `set_associative.cc` where in the enclave mode, we emulate way deflection by returning the ways allocated for the enclaves to the replacement policy (in function `getPossibleEntries`). For the indexes of the cache we use the function named `extractSet` where we replace the higher bits with the VPT entry.
- **Program:** In the server, we include the following benchmarks (with corresponding versions) for your access:
  - SPEC2017 - 1.0.2 - private
  - PARSEC - 3.0-beta-20150206 - public
  - MiBench - 1.0 - public
  - Post-Quantum Cryptography (PQC) programs: BIG QUAKE, CRYSTALS-KYBER, CFPKM, Compact-LWE, DAGS - N/A - public
- **Compilation:** During the evaluation process, you are not obligated to compile anything. If desired, we have included compilation steps for gem5 in the Section A.8 in the appendix. The artifact already possesses every dependency for the compilation which are the same as the baseline gem5 CPU simulator.
- **Transformations:** There are no transformation tools required.
- **Binary:** The main binary file for the simulations is already pre-generated (`gem5/build/X86/gem5.opt`) on the platform. The McPAT simulators used in our evaluation also have pre-compiled binaries in `area_power_estim/mcpat` and `area_power_estim/mcpat_extra_tag`. The purposes of the two versions of McPAT are described in Section A.6.2.
- **Run-time environment:** On the reviewer's side, any OS that has SSH can run remote access the environment we provide and run all simulations. For context, the environment on the server is Debian 8.
- **Hardware:** We use the gem5 CPU simulator, an open source architecture simulator which is already included in the server (we don't require any installation or any extra hardware).
- **Execution:** The only condition we have is the establishment of the SSH connection to our platform. Simulations take few hours to few days depending on the experiment.
- **Security, privacy, and ethical concerns:** There are no security implications on the reviewer-side.
- **Metrics:** For performance metrics, we consider normalized Instructions Per Cycle (IPC). As for area/power estimation, we consider mm^2 for area and Watts for power generated by McPAT, an integrated power, area, and timing modeling framework for various architectures.
- **Output:** The files and directories that contain the metrics are explained in detail in Section A.6.
- **Experiments:** The experimentation process is explained in detail in Section A.5 .
- **How much disk space required (approximately)?:** While there's no requirement on the disk-space on the reviewer's machine, the platform memory has to be kept track of.
- **How much time is needed to prepare workflow (approximately)?:** No time needed to prepare the workflow.
- **How much time is needed to complete experiments (approximately)?:** Depending on the experiments, it can take a few hours to 3 days where full system simulations (like PARSEC experiments) take days and system emulation of security benchmarks (like PQC and MiBench security programs). The McPAT estimation experiments take a few seconds to complete.
- **Publicly available (explicitly provide evolving version reference)?:** No.
- **Code licenses (if publicly available)?:** No.
- **Data licenses (if publicly available)?:** No.
- **Workflow frameworks used?:** We use custom scripts that spawn shell commands to be run concurrently. The McPAT simulations are run manually from the command line.
- **Archived (explicitly provide DOI or stable reference)?:** No.

A.3 Description

A.3.1 How to access

N/A

A.3.2 Hardware dependencies

N/A

A.3.3 Software dependencies

The only software required is Secure SHell (SSH) which we are going to use to connect to the artifact platform.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

A.4 Installation

We do not require any reviewer-side installation other than SSH (please refer to online sources for proper installation). The artifact is going to be accessed remotely. One can access our servers through:

```
ssh <username>@<dns_tag>
```

where the `username`, the DNS tag and the password are granted in the submission.

We use our platform as the artifact testing environment. First off, the reviewer is going to access the server (required information provided in artifact submission form).

Once accessed, we highly recommend that you create a `tmux` session so that in case of an unintended disconnect or any other disturbance, the experiment process can keep running independent from the remote user. To create a tmux session, the following command line can be used:

```
tmux new -s <name_of_session>
```

We have already created a session named `reviewer`. In order for a user to attach to a session, the following command line can be used:

```
tmux a -t <name_of_session>
```

To detach from a session, press "Ctrl+B" and then "Q" on your keyboard; this will send you back to the main terminal interface. When attached to a session, the user will run terminal command to run the experiments we set up. This way, if you run an experiment on the tmux session, it will keep running on the server even when you disconnect.

For PARSEC experiments, we use gem5's full system simulation for multi-threaded applications. We have already prepared the disk image and the kernel image for this simulation. The gem5 version we use requires the path to the simulator as an environment variable; so after each connection or session creation where a PARSEC-related simulation is going to be run, please set the environment variable `M5_PATH` as:

```
export M5_PATH=/home/reviewer/gem5
```

Keep in mind that you don't need to do this constantly if your are running on a prepared tmux session.

A.5 Experiment workflow

We have 2 different simulation environments. The first one is the performance results gathered from the gem5 CPU simulator which we modified to emulate Composable Cachelets.

A.5.1 Performance Results from the gem5 Simulator

Experiment Scripts Explained We have three Python scripts for each benchmark suite: `runspec.py`. `runmi.py`, and `runparsec.py`. These scripts create simulation threads for various configurations by calling the gem5 binary (`gem5.opt`). On top of that, for non-enclave simulations, we use `runnonenc.py`.

gem5 is modified to have extra convenience options to the baseline such as `l3_vpt_size` that defines the number of VPT entries. Another example is `l3_cachelet_assoc` which defines the associativity of allocated cachelets. The user will not be interacting with these options directly, but it is important to note.

SPEC2017 Enclave Experiments For SPEC2017 benchmark suite evaluations, we use `runspec.py`. It tests 14 benchmarks from the suite and tests one 10 cachelet configurations along with the baseline configuration. The script has 3 mandatory options which are `real_insts`, `warmup_insts`, and `jobs` where they denote the number of real instructions (instruction considered for performance evaluation), number of instructions considered as initialization (these are ignored) and number of concurrent experiments. The platform we provide has 48 cores and all of the experiments are required, we recommend that you initialize `-jobs` as 40. Having said that, to run 1 billion real instruction and after 1 billion warm up instructions with 40 concurrent jobs, the following command would be executed:

```
python3 runspec.py --real_insts=1000000000
    --warmup_insts=1000000000 --jobs=40
```

Unfortunately, we cannot support further customization. For spec benchmarks. Please refer to the source code of the scripts for further customization.

Security Enclave Experiments For security benchmarks (MiBench and PQC), we have `runmi.py` where it takes only the same `jobs` argument. This script runs 4 configurations (including the baseline) per 8 benchmarks we consider.

The following command is an example of security benchmark experiment with 40 concurrent jobs:

```
python3 runmi.py --jobs=40
```

PARSEC Enclave Experiments Finally, for PARSEC benchmarks, we have a similar option layout for the script (`runparsec.py` in this case) to the security benchmarks. However, since PARSEC experiments are done in full system simulation we recommend 3 concurrent jobs at maximum. There is only 1 configuration per benchmark, and we consider 5 of them. To run PARSEC benchmarks, the following command line can be run:

```
python3 runparsec.py --jobs=2
```

SPEC2017 Non-Enclave Experiments For non-enclave experiments, we considered 5 SPEC benchmarks. The script we use for these experiments has the same option layout as `runspec.py`. To run these experiments:

```
python3 runnonenc.py --real_insts=1000000000
    --warmup_insts=1000000000 --jobs=40
```

is going to be used as the command.

A.5.2 Area and Power Estimation from McPAT

The files involved in this experimental workflow are located in `/home/reviewer/area_power_estim/`, under the following subdirectories:

- `mcpat`: contains the unmodified mcpat simulator source code and binaries from the public git repository.
- `mcpat_extra_tag`: contains a version of the mcpat source and binaries that we modified to simulate additional cache tags. The source differs from the baseline `mcpat` only in the `/cacti/const.h` header file, where the value of the `EXTRA_TAG_BITS` variable was changed from 5 to 9.
- `cc_descriptions`: contains the architecture descriptions used to run the simulations.
- `cc_mcpat_final_results`: contains the area/power result files from which we extracted the estimates.

We estimated the area and power overheads for major CC components (the VPT, CFL, and extended cache tags) using the `mcpat` computer architecture simulator. We describe the configurations used for our simulations below. The results of the simulations, and the procedures for running them, are presented in Section A.6.2

Baseline processor: As a point of comparison for our area and power results, we took one of the default processor specification files (Intel Xeon processor) provided with mcpat, and modified the Caches, Register File, TLB, BTB, LSQ, ROB, and fetch/decode/issue/commit width parameters to match the architecture used elsewhere in our evaluations. This file is located at:

```
/home/reviewer/area_power_estim/cc_descriptions/
    cc_base_processor.xml
```

VPT and CFL: We observed that the components VPT and CFL components of CC are closely analogous to existing hardware components simulated by McPAT. The VPT is comparable to a register alias table (retirement RAT), while the CFL corresponds to a register free list. Thus, we used the McPAT area and power metrics for a Integer Retirement RAT and Free List as our estimates for the area and power of the VPT and CFL, respectively. When generating results for each component, we configured the simulation so that the components would have a similar size and organization to VPT and CFL described in our paper:

- For the configuration that generated the VPT results (`/home/reviewer/area_power_estim/cc_descriptions/cc_single_issue_vpt.xml`), we specified 16 architectural registers and 64 physical registers. The resulting RAT would be comparable to a 16-entry VPT in a CC system with 64 total cachelets. Because a VPT does not require multiple write ports, we also gave the processor a single instruction width, so that the rename logic has a single write port.
- To obtain a 64-entry register free list (comparble to the 64-entry CFL assumed in our area/power evaluation), we modified the baseline processor specification to have 64 physical registers. The configuration file used for the CFL results was `/home/reviewer/area_power_estim/cc_descriptions/cc_free_list.xml`

Extended Cache Tags: To estimate the area and power overhead of the additional cache tag bits required by CC, we changed the number of extra cache tag bits defined in the McPAT source code. We then recompile the simulator. Specifically, we edited the file

```
/home/reviewer/area_power_estim/mcpat_extra_tag/
    cacti/const.h
```

to change the value of the `EXTRA_TAG_BITS` variable from `5` to `9`, reflecting the number of extra tag bits needed to support up to 16 cachelets per enclave. Using the baseline processor configuration `cc_base_processor.xml`, we ran the simulation with the modified and the unmodified versions of McPAT, then took the difference between the total area and power results for each simulation to determine the area and power increase attributable to the the added tag bits.

A.6 Evaluation and expected results

A.6.1 Performance Results from the gem5 Simulator

After the simulation ends, gem5 generates output directories which contain the architecture parameters and results. In these directories there are two important files, namely `config.ini` (architectural parameters) and `stats.txt` (computational metrics). `stats.txt` contains the IPC values that we are going to inspect. The scripts we use generate such output directories with names with the prefix `m5out`. Hence, to extract the IPC values from any benchmark, we use grep commands such as:

```
grep "switch_cpus_1.commit.committed_per_cycle::mean"
    artifact_cc/m5out_*/stats.txt
```

To elaborate, this command will print out all of the IPC results. Ideally we would like to specify the configuration or the benchmark of the experiments we desire to examine. We explain below how to extract IPC values from all experiments.

SPEC2017 Enclave Outputs To extract IPC values from a specific benchmark (say `deepsjeng`), the following commands should be run (baseline case first, other cases for the latter):

```
grep "switch_cpus_1.commit.committed_per_cycle::mean"
    artifact_cc/m5out_deepsjeng_baseline/stats.txt

grep "switch_cpus_1.commit.committed_per_cycle::mean"
    artifact_cc/m5out_deepsjeng_enclave*/stats.txt
```

If we want to compare different benchmarks with the same configuration (say 4 way 8 cachelets), the following command should be run:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
    artifact_cc/m5out_*enclave_4_8/stats.txt
```

Security Enclave Outputs As an example, to extract IPC from the aes benchmark, the following command should be run:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
     artifact_cc/m5out_aes_encrypt*/stats.txt
```

MiBench related benchmarks' (blowfish, sha and aes) directories are followed by an _encrypt suffix. PQC related benchmarks do not have such suffixes. Thus, for instance, getting IPC values from the benchmark BIG QUAKE is done by:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
    artifact_cc/m5out_BIG_QUAKE*/stats.txt
```

PARSEC Enclave Outputs PARSEC benchmarks are saved with a parsec_ prefix, so to extract results from the benchmark blackscholes:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
artifact_cc/m5out_parsec_blackscholes*/stats.txt
```

Non-Enclave Outputs Non-enclave experiments have _nonenclave suffix. So, for inspecting omnetpp with 12 ways the following command should be used:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
artifact_cc/m5out_omnetpp_nonenclave_12/stats.txt
```

So to check the same benchmark for all cases:

```
grep
"switch_cpus_1.commit.committed_per_cycle::mean"
artifact_cc/m5out_omnetpp_nonenclave*/stats.txt
```

Expected Results For expected results, please refer to the main paper where every detail is already discussed in detail. Since we are using simulations, there might be slight differences to the results in the paper. However, the patterns the reviewers extract shall generate the same patterns as the ones in the paper, even though the process is not fully-deterministic.

	area mm^2 (% base)	peak W (% base)	runtime W (% base)
Base arch	45.183 (100)	70.0737 (100)	35.1191 (100)
VPT	0.00042 (0.00093)	0.0022 (0.0031)	0.0066 (0.019)
VPT $\times$ 2	0.00084 (0.0019)	0.0044 (0.0063)	0.013 (0.037)
CFL	0.019 (0.042)	0.060 (0.085)	0.057 (0.16)
Tag bits	0.36 (0.80)	0.21 (0.29)	0.11 (0.33)

Table 1: Results of McPAT simulations of CC components. Peak and runtime refer to peak dynamic and runtime dynamic, Percentages are relative to the baseline architecture (Base arch)

A.6.2 Area and Power Estimation from McPAT

Our area and power estimates indicated that the main CC hardware components impose a modest overhead in terms of processor area and power. Table 1 presents our results. Note that the same results are in the submitted paper were rounded to a different level of precision.

These results above were obtained as follows:

Baseline Architecture The area and power metrics for the baseline architecture were obtained with the following command:

```
cd /home/reviewer/area_power_estim/mcpat
./mcpat -print_level 5 -infile ../cc_descriptions/
    cc_base_processor.xml > ../
    cc_mcpat_final_results/cc_5_tag
```

The "Base arch" results shown in Table 1 come from the "Processor" section of the output file.

VPT estimates The area and power metrics for the VPT were obtained with the following command:

```
cd /home/reviewer/area_power_estim/mcpat
./mcpat -print_level 5 -infile ../cc_descriptions/
    cc_single_issue_vpt.xml > ../
    cc_mcpat_final_results/cc_single_issue_vpt_res.
    txt
```

The "VPT" results shown in Table 1 come from the "Int Retire RAT" section of the output file. Note that in the paper, we multiply the results by two to reflect the presence of two VPTs in a two core processor.

CFL estimates The area and power metrics for the CFL were obtained with the following command:

```
cd /home/reviewer/area_power_estim/mcpat
./mcpat -print_level 5 -infile ../cc_descriptions/
    cc_free_list.xml > ../cc_mcpat_final_results/
    free_list_as_64_entry_cfl.xml
```

The "CFL" results shown in Table 1 come from the "Free List" section of the output file.

Tag bit overheads The tag bit overheads were obtained with the following command:

```
cd /home/reviewer/area_power_estim/mcpat_extra_tag
./mcpat -print_level 5 -infile ../cc_descriptions/
    cc_base_processor.xml > ../
    cc_mcpat_final_results/cc_9_tag
```

The "Tag bits" results shown in Table 1 are the difference between the values in the "Processor" section of the `cc_9_tag` output file and the corresponding values in the baseline `cc_5_tag` file.

A.7 Experiment customization

We provide customization in terms of number of concurrent jobs and instruction numbers (when necessary). Yet, Due to the sheer number of the considered benchmarks, we cannot provide extensive customization for benchmark specification. However, if some benchmarks are not needed by the reviewers, they can be commented out.

A.8 Notes

If desired, the reviewers can compile gem5 (which is not necessary). To construct gem5, after entering the gem5 subdirectory by:

```
cd gem5
```

the following command should be used:

```
python2 `which scons` build/X86/gem5.opt -j 40
```

Unless any change is applied to gem5, this command will print the message:

```
scons: `build/X86/gem5.opt' is up to date.
```

Also, please make sure that only one reviewer runs a specific experiment at a time to prevent interference, since aforementioned tools/frameworks overwrite namesake files/directories.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.