# **DAGguise**

Mitigating Memory Controller Side Channels

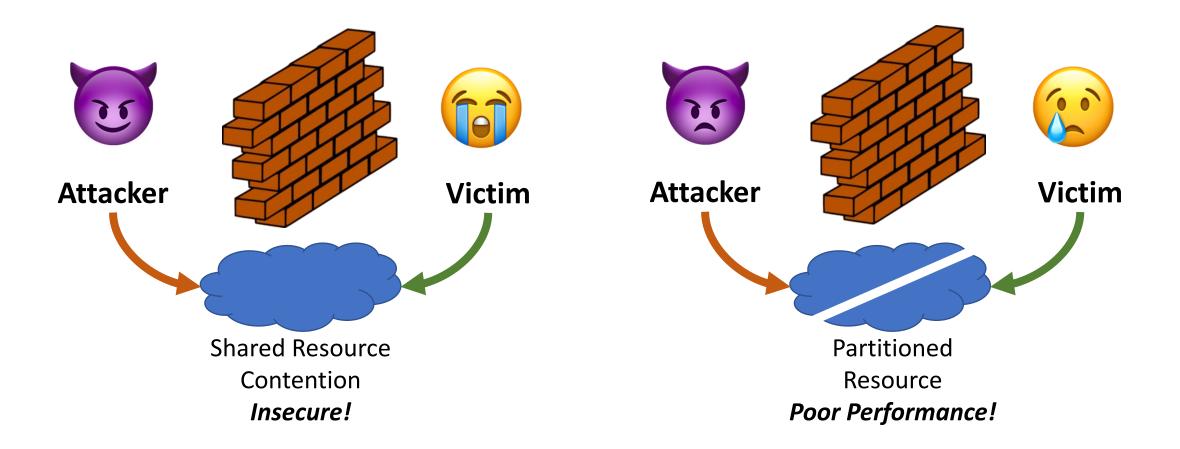
**Peter W. Deutsch\*, Yuheng Yang\***, Thomas Bourgeat, Jules Drean, Joel Emer, and Mengjia Yan

ASPLOS 2022 (Session 3A)



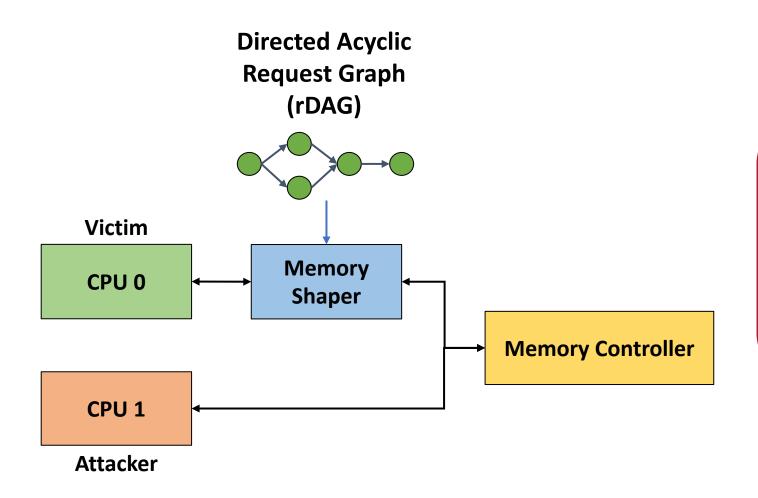


### Microarchitectural Side-Channels



**Key Defense Tradeoff:** Security vs. Performance

# **DAGguise Key Idea**



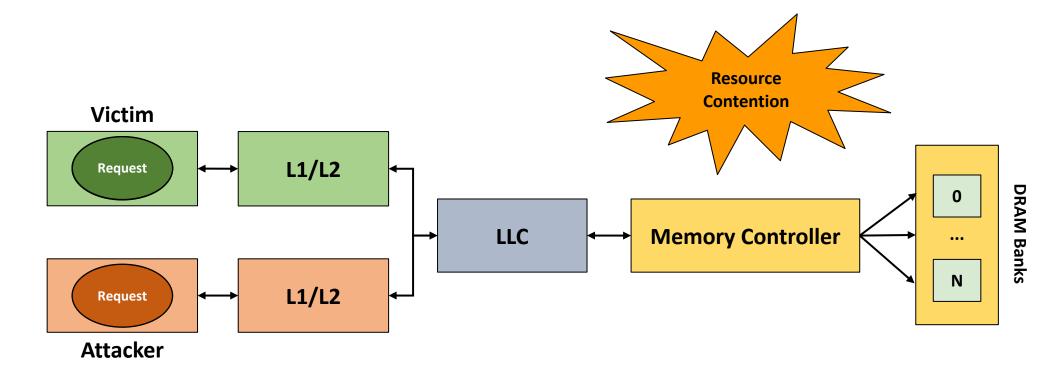
### **DAGguise achieves:**

✓ Formally-Verified Securityand✓ Good Performance

### **Outline**

- Memory Controller + Scheduler-based Side Channels
- Existing Approaches
  - Static Partitioning
  - Traffic Shaping
- DAGguise
  - Directed Acyclic Request Graphs (rDAGs)
- Security + Performance Evaluation
- Generalizability

# **Memory Controller Side Channels**



This is a class of "scheduler-based" side channels!

### Scheduler-Based Side Channels

This is the extended version of a paper that appears in USENIX Security 2021

#### Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical



Riccardo Paccagnella Licheng Luo Christopher W. Fletcher University of Illinois at Urbana-Champaign

We introduce the first tacks that leverage cont There are two challens exploit this channel. Fin connect's functioning a that can be learned by a noisy by nature and has the first challenge, we pe of the sophisticated prot the ring interconnect. W core covert channel over of over 4 Mbps from a s cross-core channel not n the second challenge, w patterns of ring contents We demonstrate our attac able EdDSA and RSA is the precise timing of key

#### 1 Introduction

Modern computers use eral heterogeneous, inte across computing units. fered significant benefit created an opportunity f croarchitectural feature of software-based cover Through these attacks. fects (e.g., timing variat resource to surreptitious nel case) or infer a vic channel case). These at ble of leaking informat ample, many cache-bas demonstrated that can le tographic keys) in cloud web browsers [40,61,77

#### Bandwidth Utilization Side-Channel on ML. Inference Accelerators

Sarbartha Banerjee The University of Texas at Austin

Shijia Wei The University of Texas at Austin shijiawei@utexas.edu

Prakash Ramrakhyani ARM Research prakash.ramrakhyani@arm.com

Mohit Tiwari The University of Texas at Austin tiwari@austin utexas edu

Abstract—Accelerators used for machine learning (ML) infer-

ence provide great perfe confidential model in it attacks is critical in har practice. Data and memproposed to defend again In this paper, we demo interface between accele a side-channel for leaking This side channel is inde even in the presence of da can be monitored throug contention from an on-cl

Deep learning mode domain-specific comput inference accelerators in (NPU) are being develop academia [8], [11], [16]. system-on-chip (SoC) Inference-as-a-service providers like Amazon on ML accelerators. Th host trained models on c confidential user data lil

data like disease classifi From the security pe the cloud provider to parameters as well as t [78], [20] show how kr used to steal a victim's !

similar accuracy. An at

2019 IEEE Symposium on Security and Privacy

#### Port Contention for Fun and Profit

Alejandro Cabrera Aldaya\*, Billy Bob Brumley<sup>†</sup>, Sohaib ul Hassan<sup>‡</sup>, Cesar Pereida García<sup>‡</sup>, Nicota Tuveri<sup>‡</sup> \*Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba Tampere University, Tampere, Finland

Abstract-Simultaneous Multithreading (SMT) architectures sets, machine learning techniques, not reverse engineering are attractive targets for side-channel enabled attackers, with sechniques. their inherently broader attack surface that exposes more per physical core microarchitecture components than cross-core attacks. In this work, we explore SMT execution engine sharing as a side-channel leakage source. We target ports to stacks of execution units to create a high-resolution timing side-channel due to port contention, inherently stealthy since it does not depend on the memory subsystem like other cache or TLB based attacks. Implementing our channel on Intel Skylake and Kabs Lake architectures featuring Hyper-Threading, we mount an end-to-end attack that recovers a P-384 private key from add and double operations during scalar multiplication. We an OpenSSI-powered TLS server using a small number of then process the signal using various techniques to clean the repeated TLS handshake attempts. Furthermore, we show that traces targeting shared libraries, static builds, and SGX enclaves are essentially identical, hence our channel has wide target

1. INTRODUCTION

To demonstrate PORTSMASH in action, we present a complete end-to-end attack in a real-world setting attacking the NIST P-384 curve during signature generation in a TLS server compiled against OpenSSL 1.1.0h for crypto functionality. Our Spy program measures the port contention delay while executing in parallel to ECDSA P-384 signature generation. creating a timing signal trace containing a noisy sequence of signal and reduce errors in the information extracted from each trace. We then pass this partial key information to a recovery phase, creating lattice problem instances which ultimately

yield the TLS server's ECDSA private key. We extend our analysis to SGX, showing it is possible to DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks

Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz and Stefan Mangard Graz University of Technology, Austria

often co-located or Thus, preventing in crucial. While the tion, shared hardwa ory bus, can leak se sons, shared memo abled. Furthermore. CPU. In this setting a slow cross-CPU o known. In contrast, channel as well as

In cloud computi

across processors a build these attacks. present two n of memory ac inks. One us ne other runs ed. Using this that is share our attacks v First, we bu o 2 Mbps, w faster than it ld a side-char locate and m now using the and in parti on DDR4.

the same p

#### SMoTherSpectre: Exploiting Speculative Execution through Port Contention

Atri Bhattacharyya' Alessandro Sorniotti

IBM Research - Zurich

Alexandra Sandulescu IRM Research - Zurich

Babak Falsafi'

EPFL.

Anil Kurmus

CCS'18, October 15-19, 2018, Toronto, ON, Canada

#### ABSTRAC

are prone to a applications, i code, may be that leverages регосимога (SA victim process world gadgets locates 5MoTi will their new Yorks mation, Finall the OpenSSH

the plaintext t

CCS CONG

· Security an

KEYWORI

attack; micros

troduction

Session 10D: VulnDet 2 + Side Channels 2

#### Rendered Insecure: GPU Side Channel Attacks are Practical

Matthias Neugschwandtner\*

IBM Research - Zurich

Mathias Payer'

Hoda Naghibijouybari University of California, Riverside hnagh001@ucr.edu

Zhiyun Qian University of California, Riverside zhiyunq@cs.ucr.edu

Graphics Processing Units (GPUs) are commonly integrated with computing devices to enhance the performance and capabilities of graphical workloads. In addition, they are increasingly being integrated in data centers and clouds such that they can be used to accelerate data intensive workloads. Under a number of scenarios the GPU can be shared between multiple applications at a fine granularity allowing a spy application to monitor side channels and attempt to infer the behavior of the victim. For example, OpenGL and WebGL send workloads to the GPU at the granularity of a frame, allowing an attacker to interleave the use of the GPU to measure the side-effects of the victim computation through performance counters or other resource tracking APIs. We demonstrate

Ajaya Neupane University of California, Riverside ajaya@ucr.edu

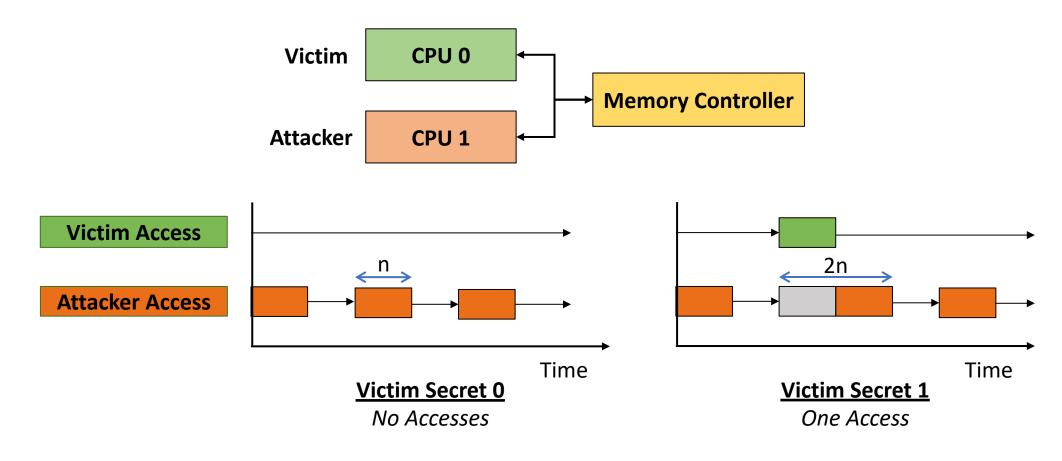
Nael Abu-Ghazaleh University of California, Riverside nael@cs.ucr.edu

#### 1 INTRODUCTION

Graphics Processing Units (GPUs) are integral components to most modern computing devices, used to optimize the performance of today's graphics and multi-media heavy workloads. They are also increasingly integrated on computing servers to accelerate a range of applications from domains including security, computer vision computational finance, bio-informatics and many others [52]. Both these classes of applications can operate on sensitive data [25, 31, 57] which can be compromised by security vulnerabilities in the

Although the security of GPUs is only starting to be explored. several vulnerabilities have already been demonstrated [46, 49, 55, 58, 63, 71, 74]. Most related to this paper, Luo et al. demonstrated a

## **Timing Attack Example**



The attacker uses its own latencies to leak information!

## **Static Partitioning in Time**

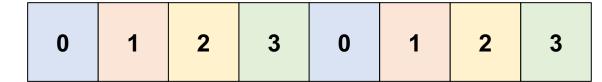
### Use a Round Robin, No-Skip Arbitration Policy

Security
Domain 0

Security
Domain 1

Security
Domain 2

Security Domain 3



**Slot Allocation Timeline** 

**√** Secure

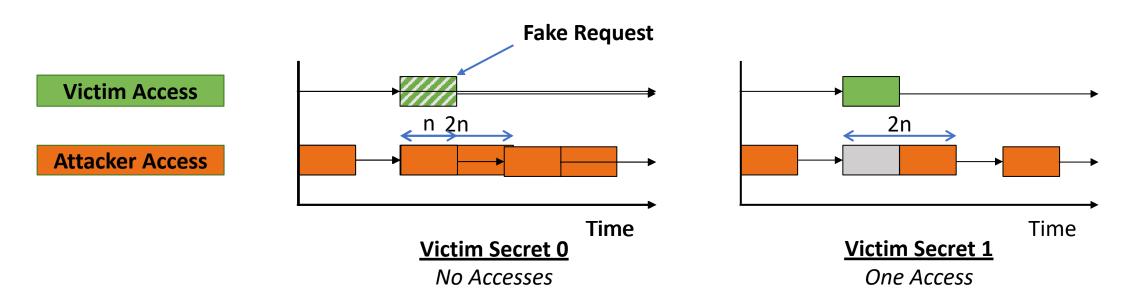
Static partitioning, no leakage

X Bad Performance

Poor bandwidth utilization!

# **Traffic Shaping**

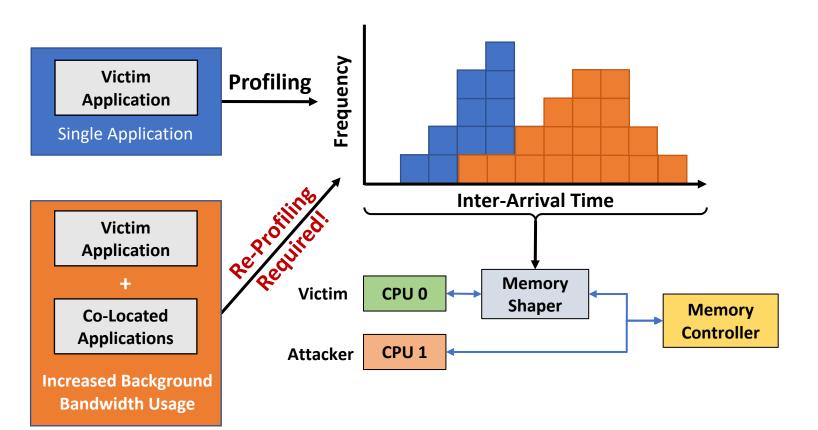
**Shaping Strategy:** Delay victim's existing requests and add fake requests



How do we do this for real applications without significant costs?

# Camouflage's Traffic Shaping Strategy

Shape memory requests to a secret-independent timing distribution



### **Good Performance**

Dynamic sharing of the memory controller

### X Insecure

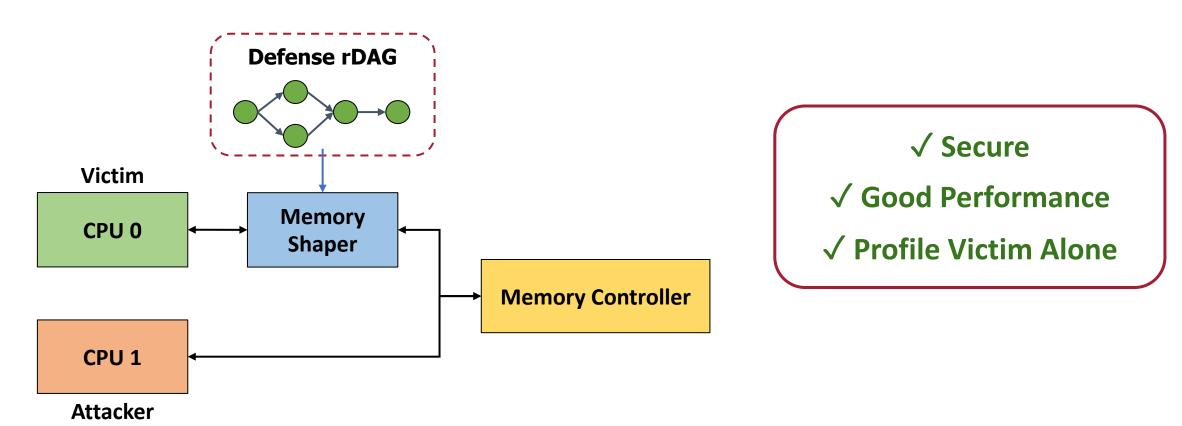
Ordering <u>or</u> bank information can reveal the secret

### **X** Expensive Profiling

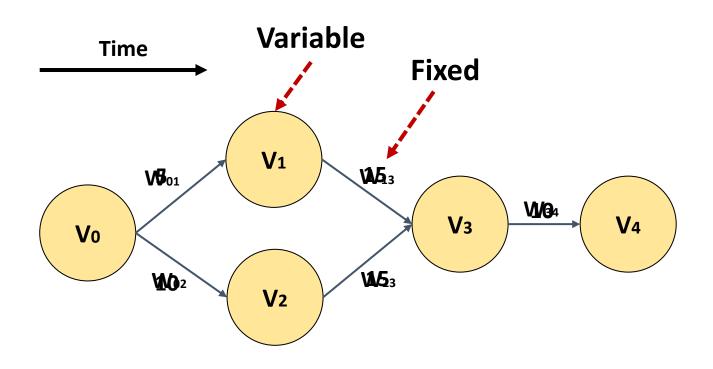
Ideal shaping distribution depends on co-running applications

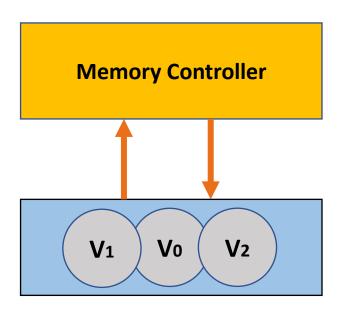
# DAGguise's Traffic Shaping Strategy

Shape memory requests to a secret-independent Directed Acyclic Request Graph (rDAG)



## **Directed Acyclic Request Graphs**





### <u>Vertices</u>

Memory requests with *variable* latency

### <u>Edges</u>

Dependencies between memory requests with *fixed* latency

## Why shape requests to an rDAG?

### ✓ Security

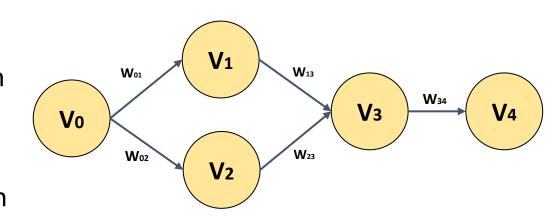
- Shaping to a secret-independent defense rDAG makes victim request patterns indistinguishable
- Defense rDAGs are public and are the only thing an attacker can recover

### **✓** Performance

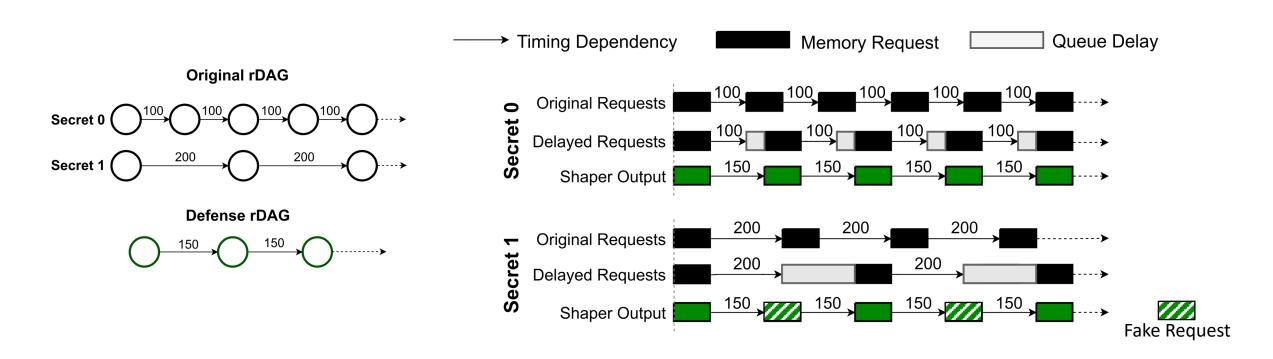
 Allows for dynamic sharing of memory resources in the memory controller

### ✓ Profiling Cost

Does not require knowledge of co-located applications

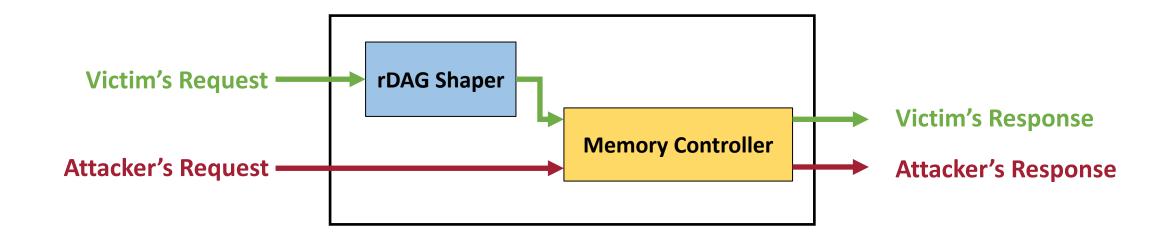


# **Simple Shaping Example**



The shaper output is always the same, no matter the secret!

# **Indistinguishability Property**



The attacker's observations should be independent from victim's request pattern

# **Indistinguishability Property**

- Attacker's observation is independent from victim's request pattern
  - Given an attacker's request pattern, the attacker has an identical observation when contending with **ANY** victim's request pattern
  - This holds for ANY attacker's request pattern

### **Attacker's Observations when Contending with Victim**

Victim Request Patterns  Attacker Request Patterns	Α	В	C	•••
X	Attacker's Response Pattern X			

### Formalization & Verification

• Formalize the indistinguishability property using state transitions

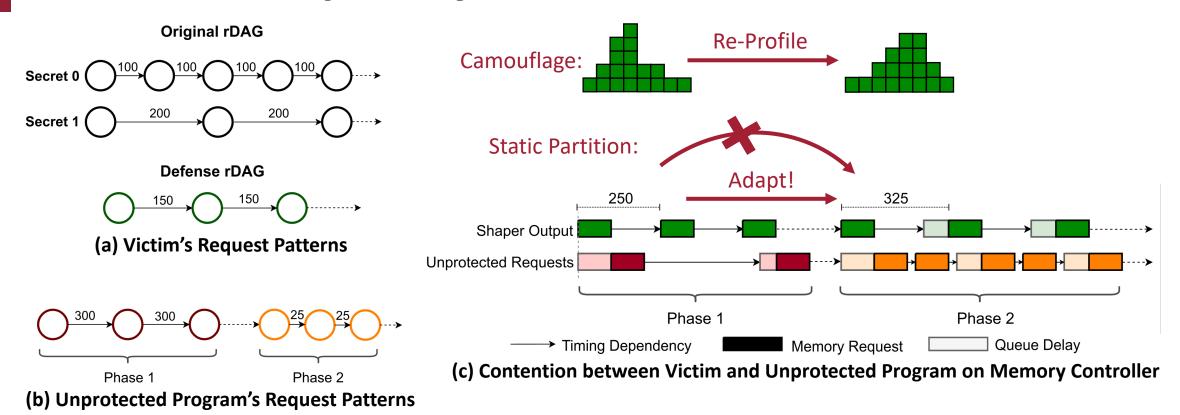
$$P(S_0,n) := \forall \operatorname{Req}_{Tx}, \operatorname{Req}'_{Tx}, \forall \operatorname{Req}_{Rx}$$

$$\text{if } S_0 \xrightarrow{\operatorname{Resp}_{Tx}, \operatorname{Resp}_{Rx}} S_n \text{ and } S_0 \xrightarrow{\operatorname{Resp}'_{Tx}, \operatorname{Resp}'_{Rx}} S'_n$$

$$\text{then } \operatorname{Resp}_{Rx} = \operatorname{Resp}'_{Rx}$$

- Verification with *Rosette*:
  - First k cycles: symbolic execution
  - Arbitrary cycles: k-induction

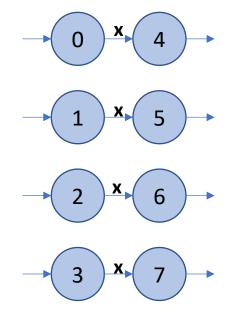
## rDAG Adaptivity



rDAG's adaptivity allows for better bandwidth utilization!

# **Offline Profiling Step**

- Not for security, any secret-independent rDAG ensures security
- Low profiling cost
  - Victim is profiled alone
  - Reduce search space by finding parameters for an rDAG template



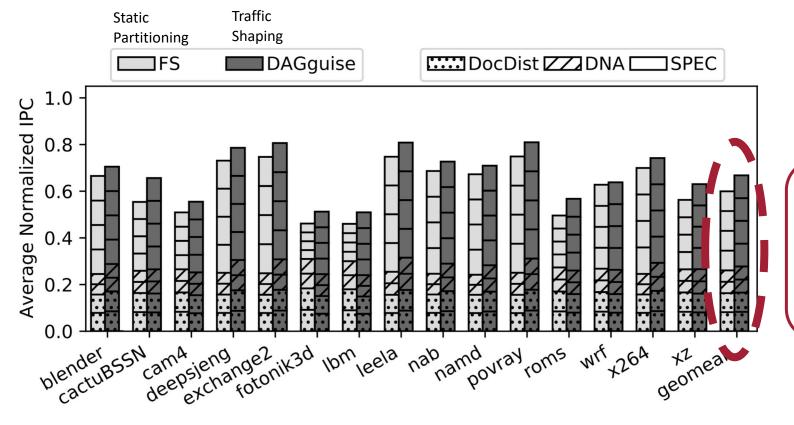
**4-Parallel rDAG Template** 

## **Experimental Setup**

- Simulator: gem5 and DRAMSim2
- Architectural Specifications:
  - 2 and 8 out-of-order CPU cores
  - 32KB L1i/d, 256kB L2, 1MB/core L3
- Evaluated Configurations:
  - DAGguise
  - Fixed Service (Bank Triple Alternation)
  - Baseline
- Evaluated Applications:
  - Unprotected SPEC benchmark(s) co-running alongside DAGguise protected application(s)



# **Experimental Results**

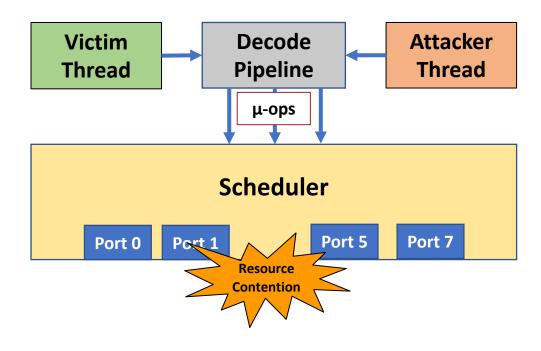


DAGguise's improves performance for both protected *and* unprotected applications!

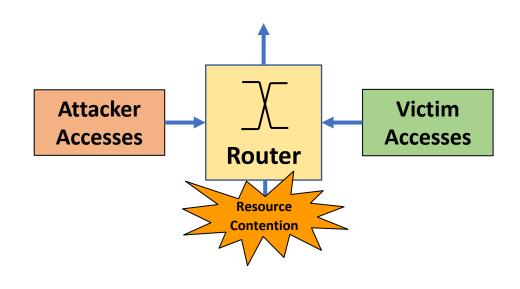
DAGguise achieves a 12% performance improvement over Fixed Service in an 8-CPU system

## **DAGguise Generalization**

### **SMT Contention**



### **Network on Chip Contention**

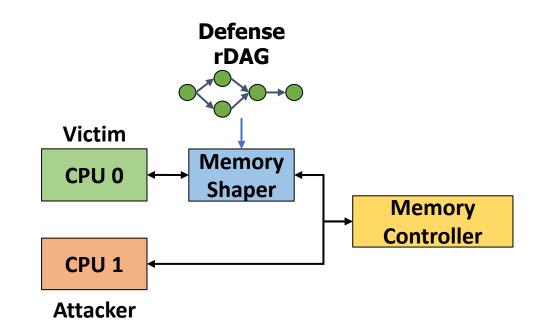


## More in the Paper

- Implementation details of DAGguise shaper
- Formal security verification using symbolic execution and k-induction
- Detailed rDAG offline profiling process
- More performance and area overhead evaluation
- Generalizations to other scheduler-based side channels (e.g. port contention)

### Conclusion

- DAGguise
  - A memory traffic shaper which:
    - Completely eliminates data leakage
    - Allows for dynamic contention
    - Requires only simple profiling
- rDAGs
  - A *general* and *adaptive* request representation
- A formal model of correctness using *Rosette*
- A generalized scheduler-based attack mitigation framework



# **DAGguise**

### Mitigating Memory Controller Side Channels

Peter W. Deutsch

pwd@mit.edu

Thomas Bourgeat bthom@mit.edu

Jules Drean drean@mit.edu

**Yuheng Yang** 

yuhengy@mit.edu

Joel S. Emer jsemer@mit.edu

Mengjia Yan mengjiay@mit.edu



