VERNAM Group

Security & Privacy @ WPI

A Faster and More Realistic *Flush+Reload* Attack on AES

Berk Gulmezoglu, Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth and Berk Sunar

COSADE-2015
13.04.2015

WPI

OUTLINE

- INTRODUCTION
- CACHE SIDE CHANNEL ATTACKS
- ATTACK DESCRIPTION
- EXPERIMENT SETUP
- RESULTS
- CONCLUSION

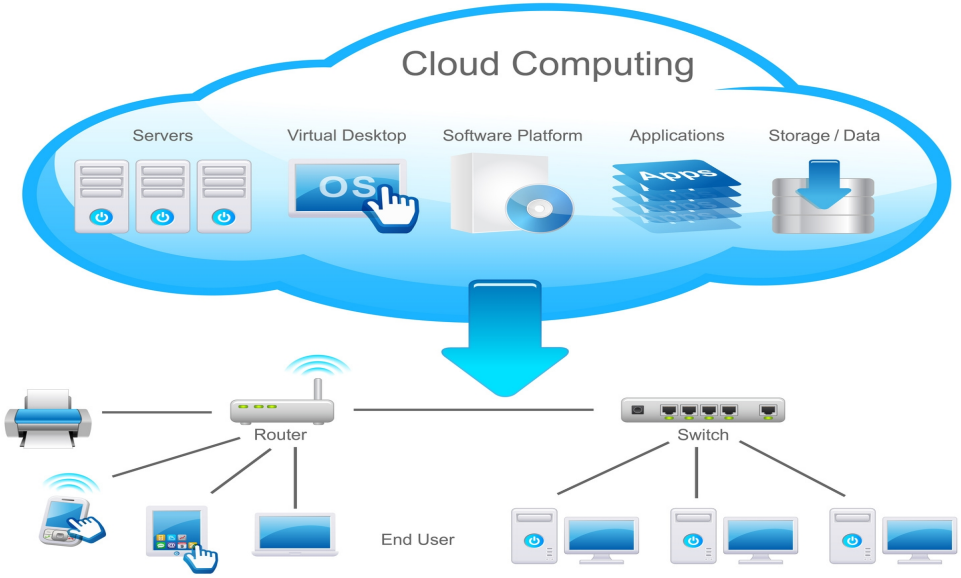
INTRODUCTION

Microsoft | Cloud Services

- Cloud computing and virtualization

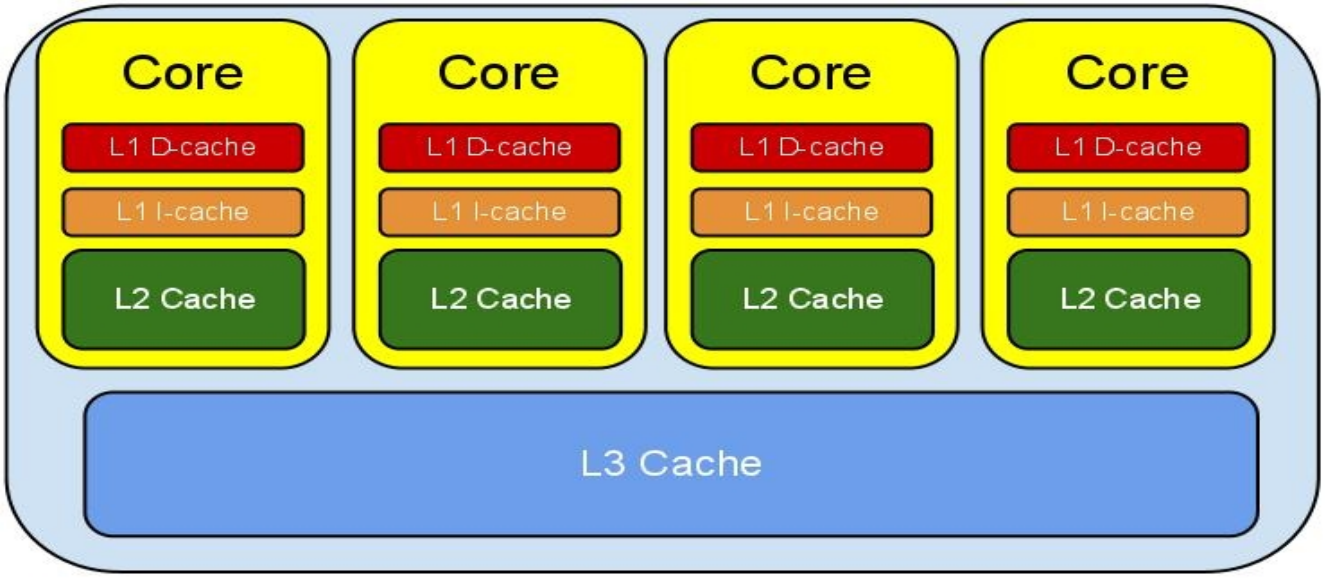
- IBM, Amazon, Microsoft, Oracle
- Threats to commercial clouds at software level

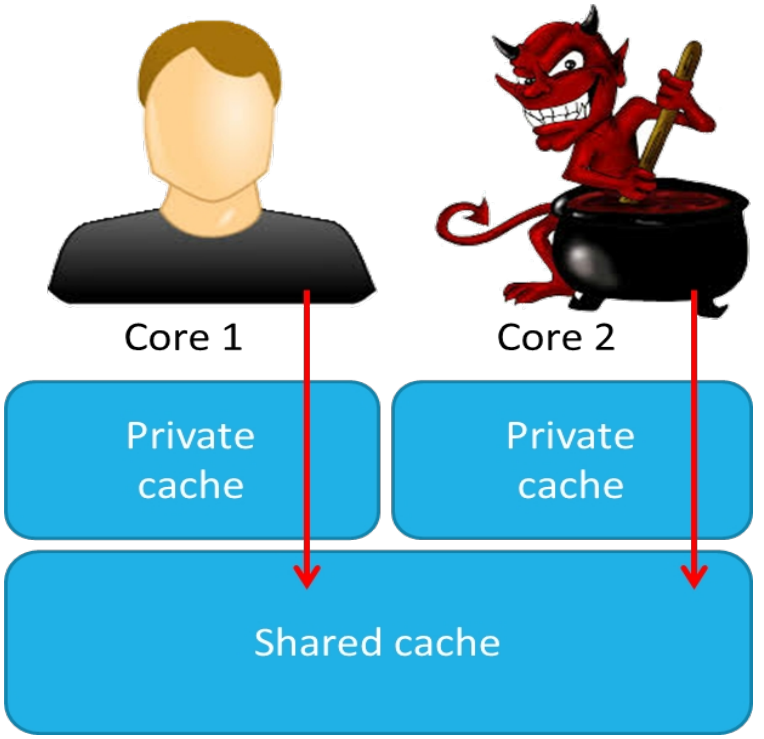
- Ristenpart et al. → Co-location
- Yarom et al. → Flush+Reload attack on RSA
- Irazoqui et al. → Flush+Reload attack on AES



CACHE SIDE CHANNEL ATTACKS

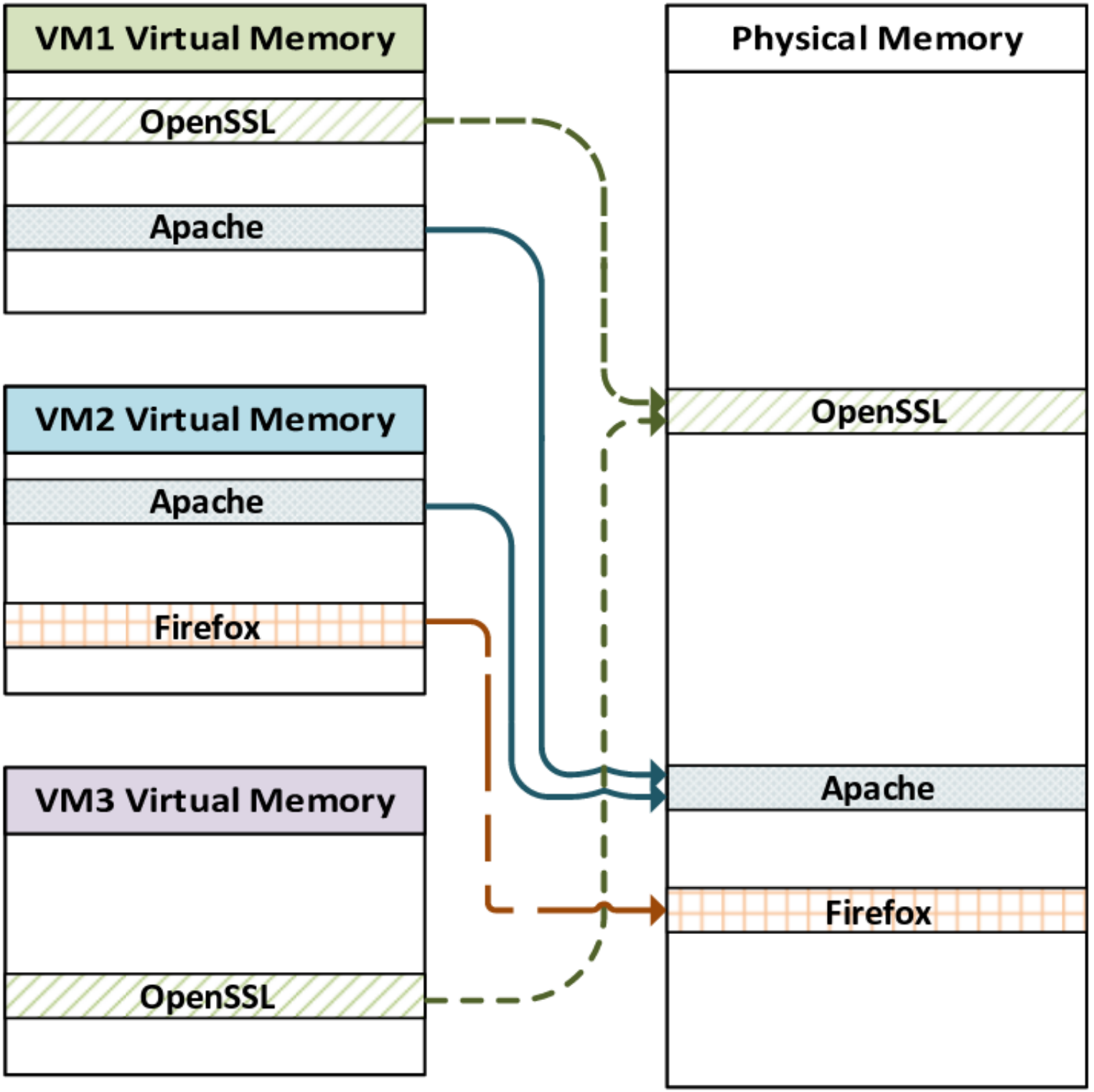
- Caches → faster memories
- Microarchitectual leakages from time variations
- Usage of side channel attacks to extract information
- L3 cache → covert channel





MEMORY DEDUPLICATION

- OS memory optimization technique
- Only a single copy of a data in the memory
- VMM  checking hash value & bit-by-bit comparison
- Applicable to shared libraries
- Transparent Page Sharing (TPS)
- Kernel Samepage Merging (KSM)

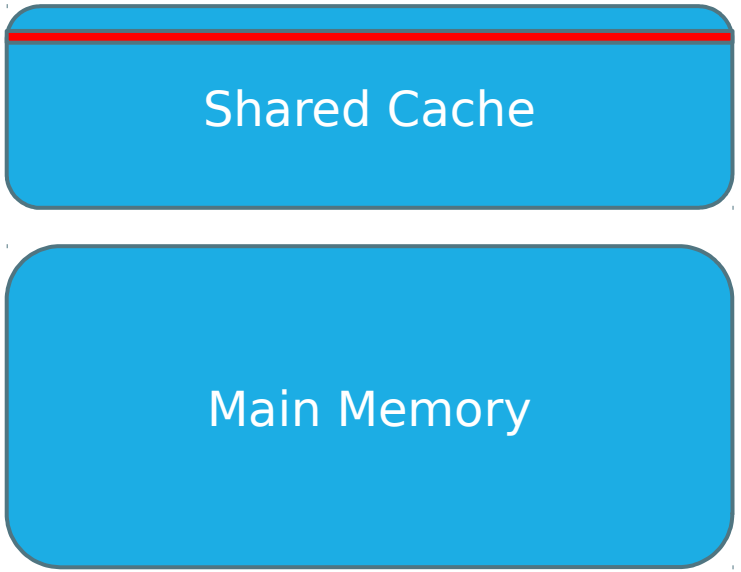

VM1 Virtual Memory
OpenSSL
Apache
VM2 Virtual Memory
Apache
Firefox
VM3 Virtual Memory
OpenSSL
Physical Memory
OpenSSL
Apache
Firefox

FLUSH AND RELOAD ATTACK

- Low noise access driven attack
- Exploit shared memory pages and deduplication
- Steps:

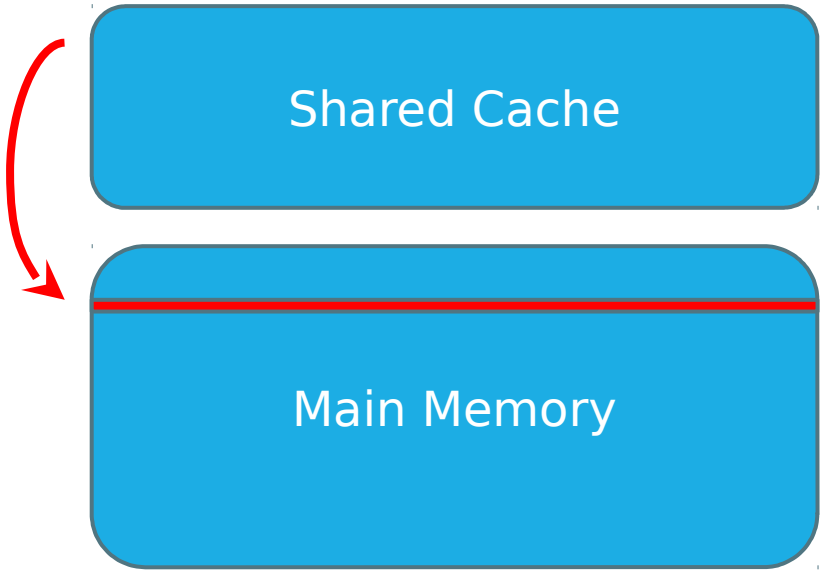
1) Flush desired memory lines
2) Wait until detecting the victim runs AES encryption
3) Flush the last round T-table entries
4) Reload the memory lines

Example

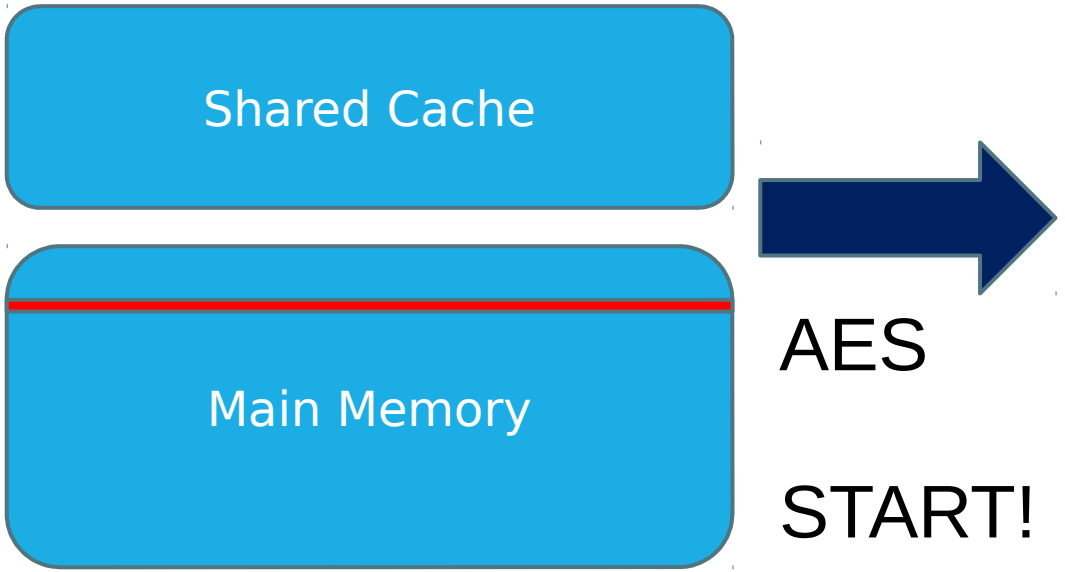


Example

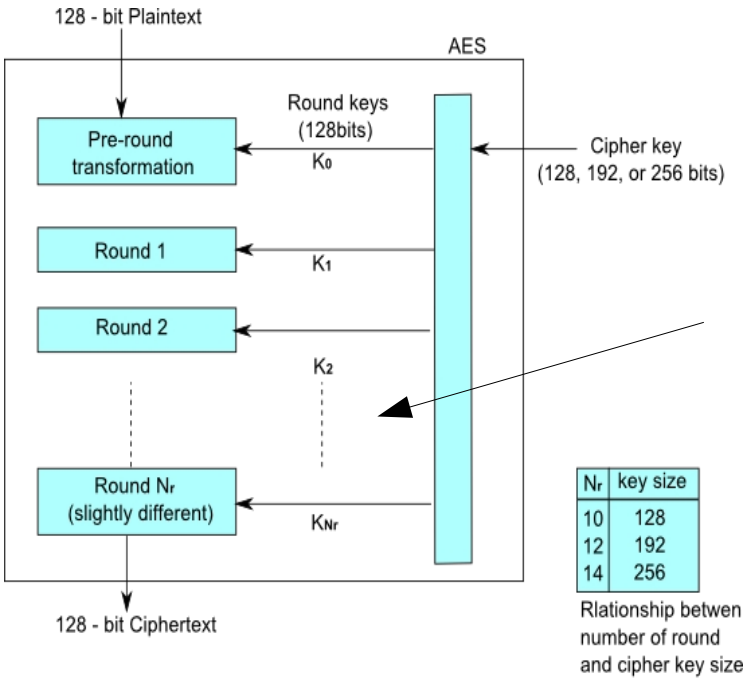
FLUSH!



Example

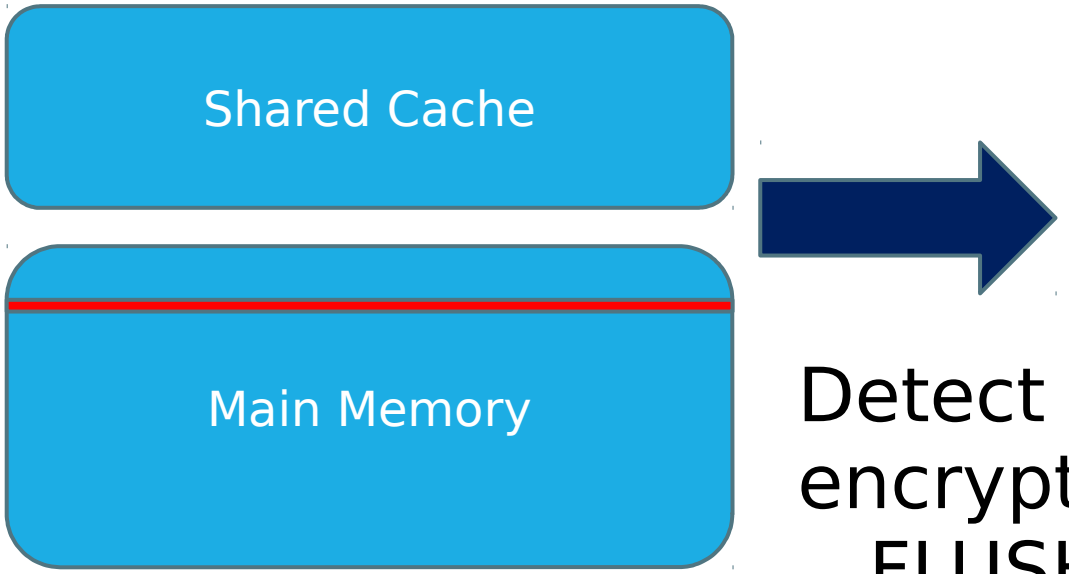


Example



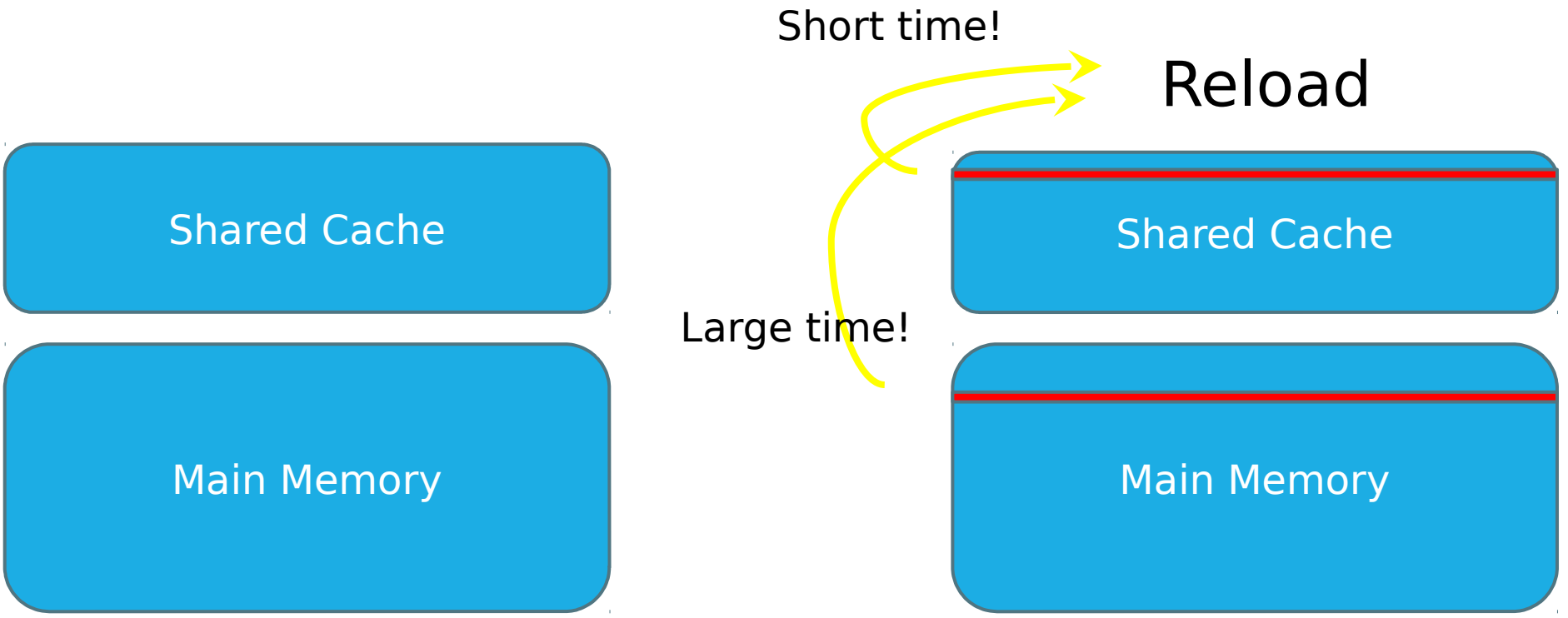
General design of AES encryption cipher

FLUSH



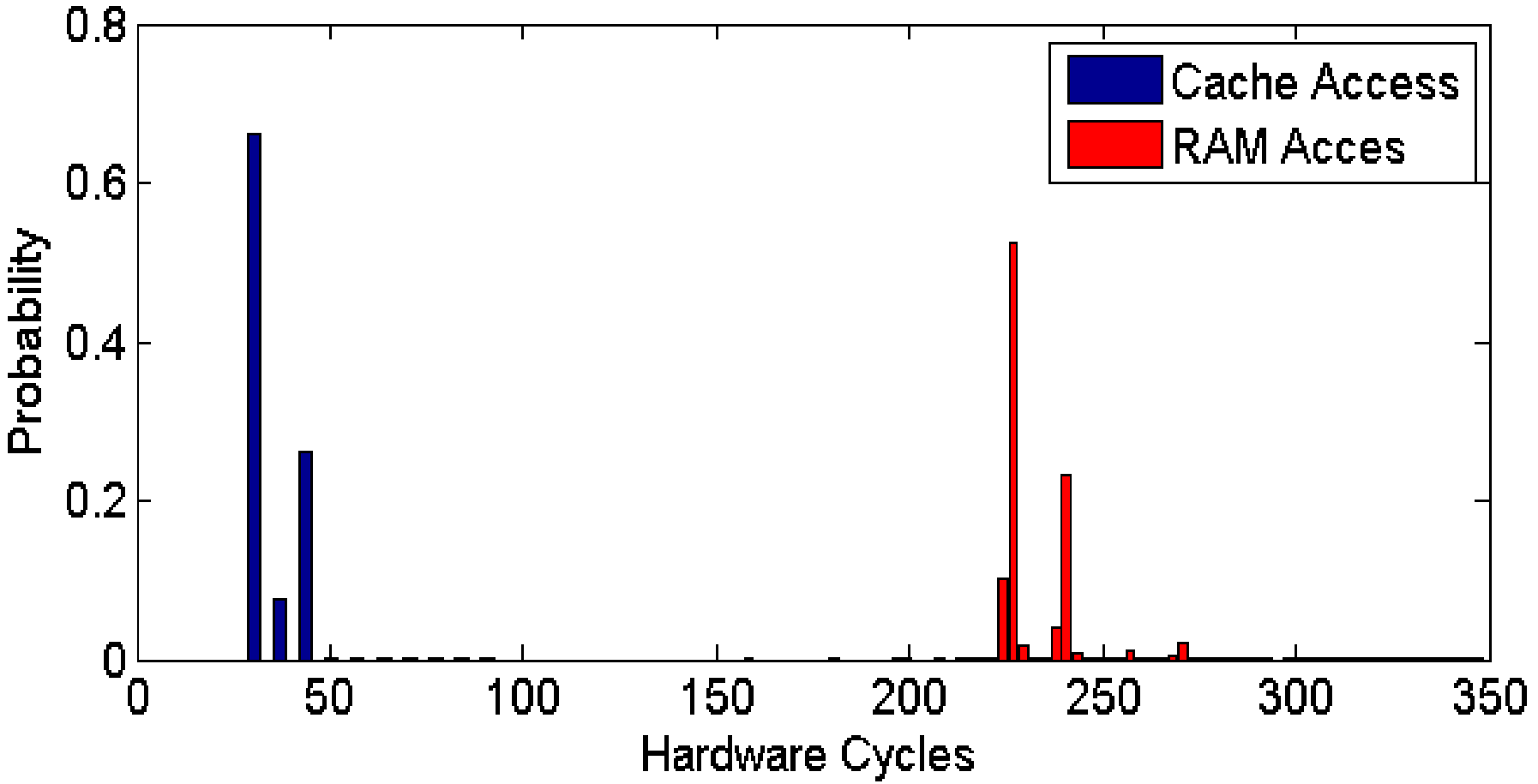
Detect AES encryption FLUSH!

Example


Short time!
Reload
Shared Cache
Large time!
Shared Cache
Main Memory
Main Memory

CACHE SIDE CHANNEL ATTACKS

- Timings for RAM and L3 Cache Access



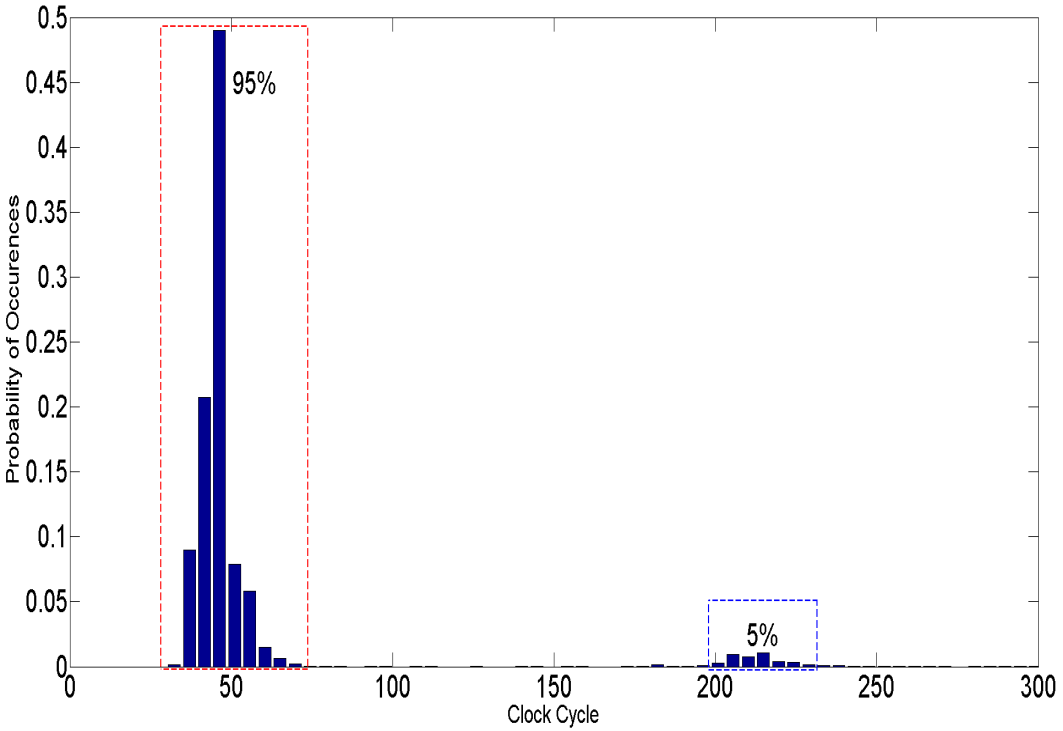
ATTACK DESCRIPTION

- **A single cache line attack on AES**
  - Monitor one of the last round T-tables
  - Collect <c,t> pairs
  - n T-table entries **T** known to *adversary*
  - Ciphertext byte C_i
  - n T-table outputs S_i
  - $c_{i,j} = k_i \oplus s_{i,j}$

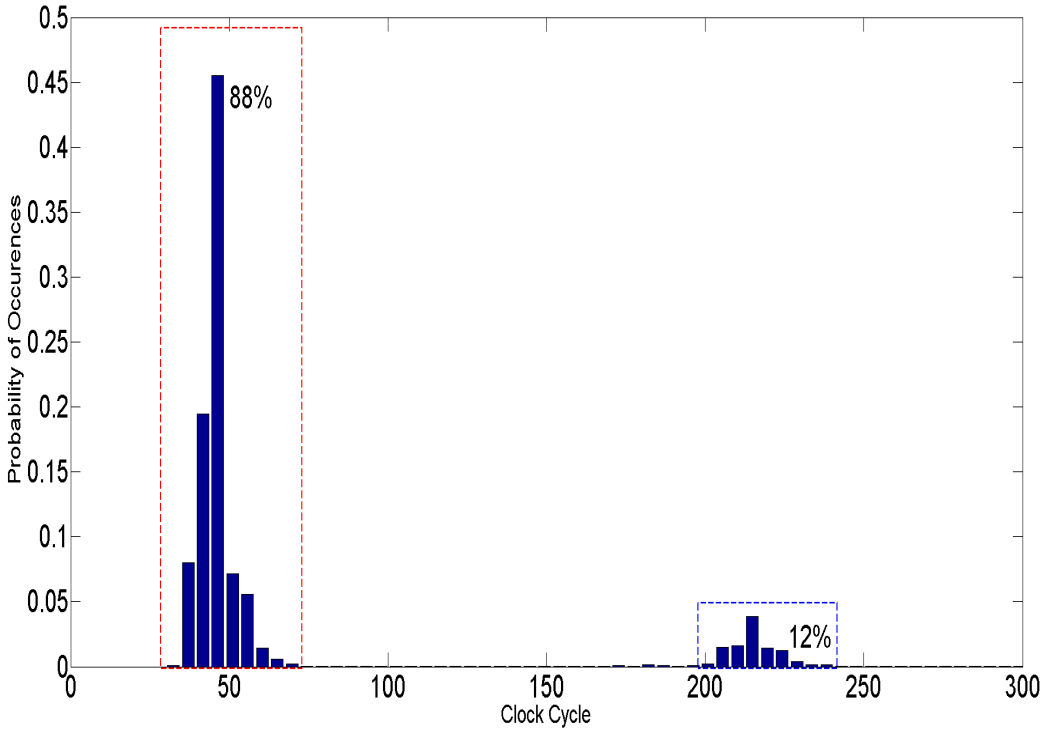
IDEAL CASE

- If $s_{i,j} \in T$ → **ACCESS** → H_0 **100%**
- If $s_{i,j} \notin T$ → **NO ACCESS** → H_1 **92%**
- For AES-128 in OpenSSL, n=16 and l=40 per T_j

  Pr[no access to T_j]= $(1-n/256)^l$



H_0



H_1

Distinguishers for the AES attack

- Miss counter based Distinguisher

- Count and the compare the relative counters of the memory block misses
- t > 130 clock cycle ⟹ miss (1)
- t < 130 clock cycle ⟹ hit (0)

$$\mathcal{D}_{miss\_ctr} = \arg\max_{\hat{k}} \left(\overline{ctr}_{H_1} - \overline{ctr}_{H_0} \right)$$

- Difference of means Distinguisher
  - Approximates the means of two distributions in cycles

$$\mathcal{D}_{means} = \arg\max_{\hat{k}} \left(\overline{\tau}_{H_1} - \overline{\tau}_{H_0} \right)$$

- Variance based Distinguisher
  - Compute the difference of variances in cycle square

$$\mathcal{D}_{vars} = \arg\max_{\hat{k}} \left(\operatorname{var} \tau_{H_1} - \operatorname{var} \tau_{H_0} \right)$$

ATTACK SCENARIOS

- *Fully Synchronous Attack (FSA)* → Original attack with synchronization
- *Semi Synchronous Attack (SSA)* → Improved version of FSA by detecting the AES encryption and flushing the T table blocks during the AES execution between rounds
- Asynchronous Attack (ASA) → No synchronization, true ciphertext only attack

✔ **More realistic attack scenario!**

EXPERIMENT SETUP

1) Native Execution: Encryption and the attacker on a native Ubuntu 12.04 LTS version. Minimal noise.

2) Cross-VM Execution: Ubuntu VMs, Vmware ESXI 5.5 baremetal hypervisor.

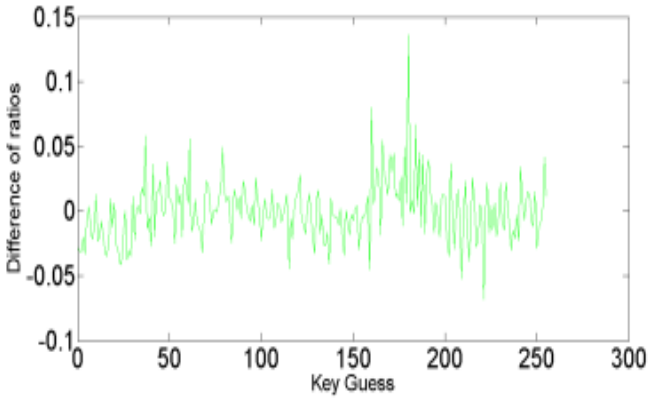
- **RDTSCP** instruction to measure the timings

↙ Not emulated by VMM executed directly

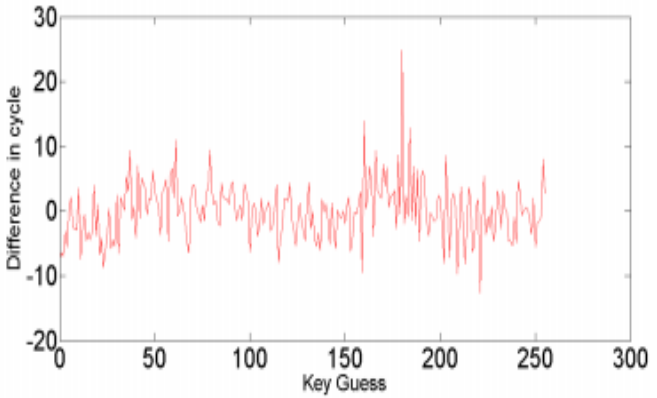
- **CLFLUSH** instruction to flush cache line

RESULTS

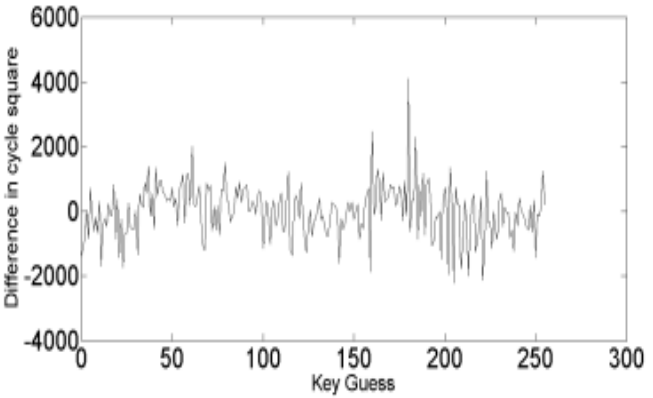
- Native Execution
  - Comparison of the scores of key guesses in the natively executed FSA scenario for three different distinguishers. (10000 traces)



(a) Ctr Dist.



(b) Mean Dist.

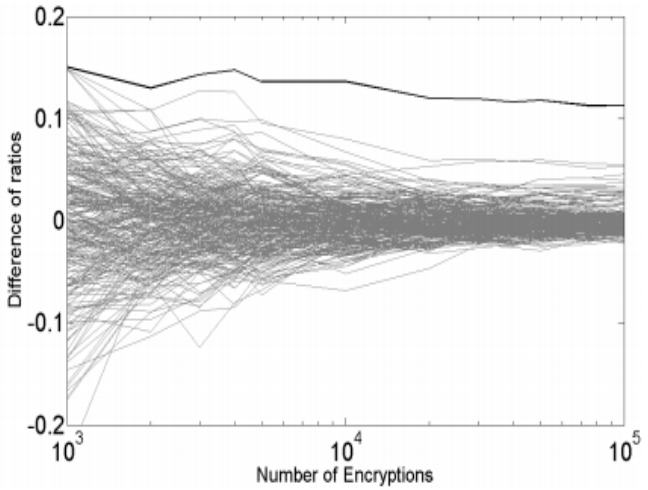


(c) Var. Dist.

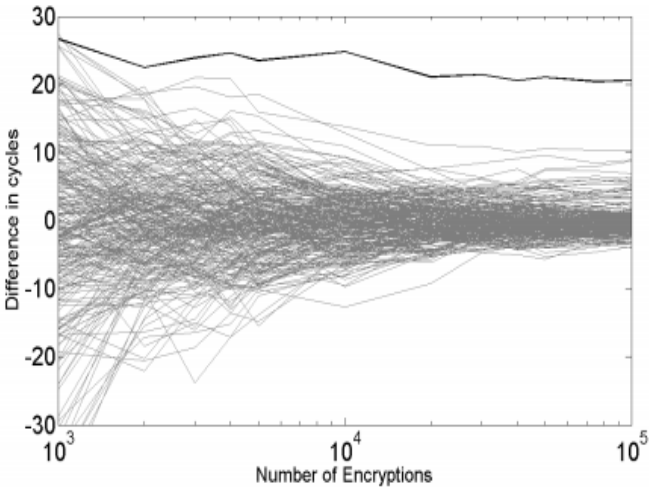
- Distribution of cache accesses vs memory accesses

Attack Scenarios	$\mathbf{H_0}$		$\mathbf{H_1}$	
	Cache	Memory	Cache	Memory
Ideal case	100%	0%	92%	8%
FSA	99%	1%	97%	3%
SSA	95%	5%	88%	12%
ASA	97%	3%	96%	4%

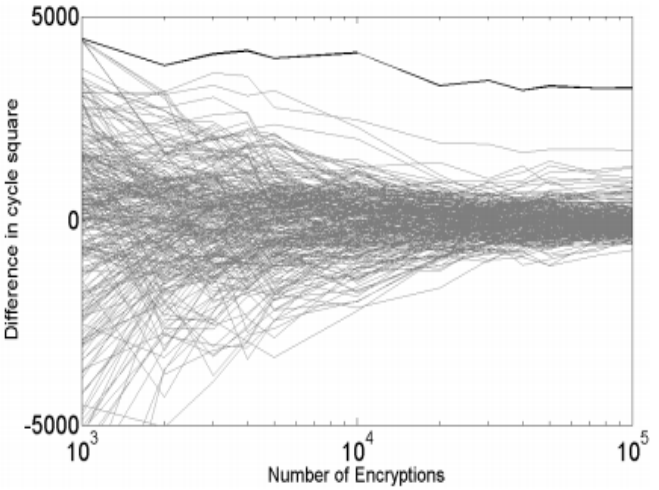
- ASA



(a) Ctr Dist.

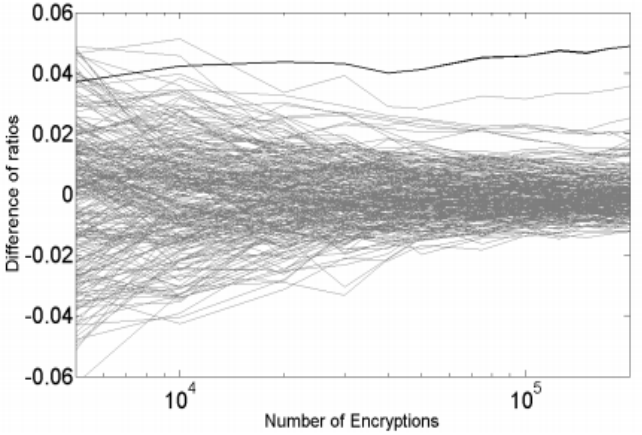


(b) Mean Dist.

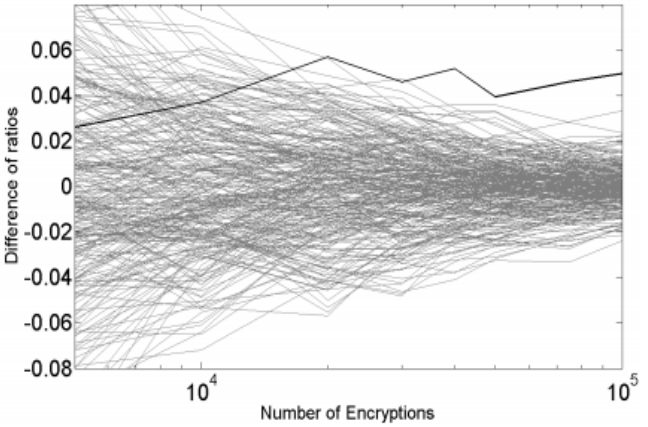


(c) Var. Dist.

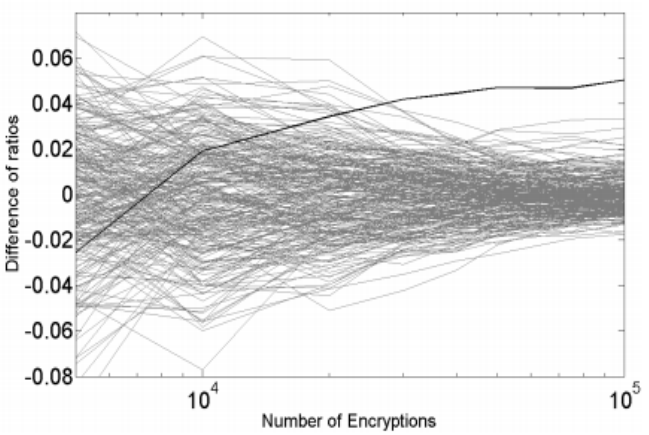
- Cross-VM execution
  - Miss-counter distinguisher



(a) FSA

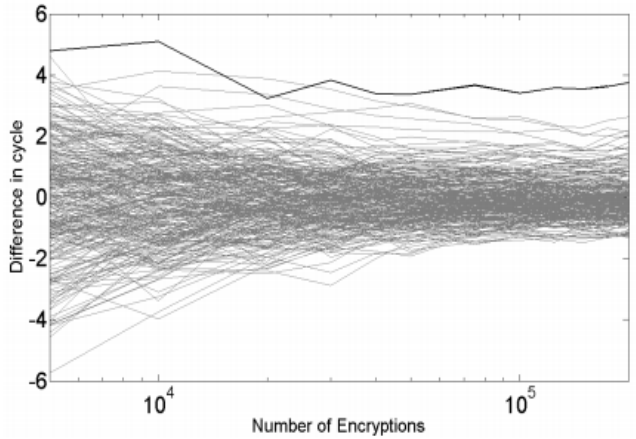


(b) SSA

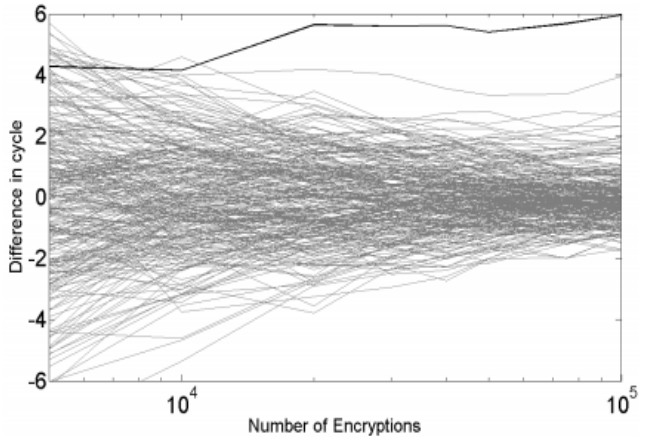


(c) ASA

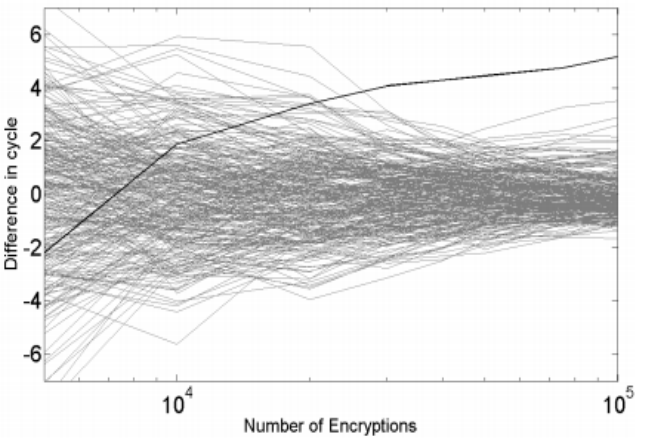
  - Means distinguisher



(d) FSA



(e) SSA



(f) ASA

	NATIVE	CROSS-VM
SSA	3000	10000
FSA	25000	30000
ASA	30000	30000

CONCLUSION

- Flushing during the AES execution  Lower noise
- More realistic attack scenario (No synchronization)
- Only 15 seconds attack
- New attack scenarios
- Different data analysis for key recovery


?
Q
&
Á
!


THANK
YOU