CSE 484 : Computer Security and Privacy

(More) Side Channel Attacks

Winter 2021

David Kohlbrenner

dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- **Homework 3 due** today
- **Last extra credit** reading due Thursday
  - No late days
- **Lab3 due** Friday
- **Final project due** 03/16
  - No late days
  - Make sure you:
    - Include references
    - Include at least one legal/ethics discussion slide
    - Create original content
    - Go beyond class materials (if it's a topic we also covered)

Admin

- **Final day?**
  - Pollev.com/dkohlbre

Course Eval

- Please fill out the course evaluation!
  - https://uw.iasystem.org/survey/236212
  - Or check email

Side-channels: conceptually

- A program's implementation (that is, the final compiled version **+ hardware**) is different from the conceptual description

- Side-effects of the difference between the implementation and conception can reveal unexpected information
  - Thus: Side-channels

Cache side-channels

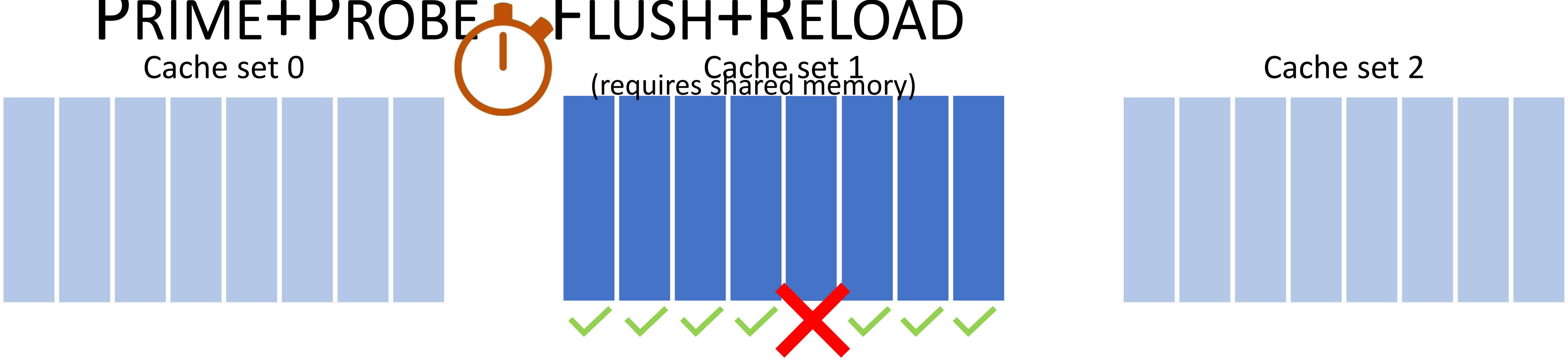
- **Idea**: The cache's current state implies something about prior memory accesses

- **Insight**: Prior memory accesses can tell you a lot about a program!

Many thanks to Craig Disselkoen for the animations.

Pre-Attack	Active Attack	Analysis
Timing threshold Eviction set	Prime targeted set → Wait → [Timed] Prime targeted set Victim accesses targeted set	Victim access if time > threshold

PRIME+PROBE → FLUSH+RELOAD

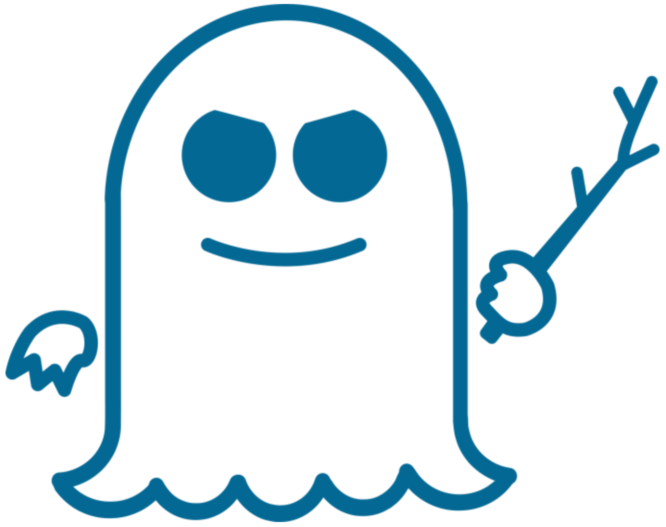


Pre-existing data | Attacker's data | Victim's data

FLUSH + RELOAD

- Even simpler!
- Kick line L out of cache
- Let victim run
- Access L
  - Fast? Victim touched it
  - Slow? Victim didn't touch it

Spectre + Friends



- First reported in 2017
- Disclosed in 2018
  - https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html
- Novel class of attack: speculative execution attacks
  - Aka: Spectre-class attacks
- (Academic paper published 2019… long story)

Two pieces of background

- Cache attacks (last week)

- Speculative execution (right now!)

Speculative Execution (the fast version)

- All modern processors are capable of speculative execution
- How much, in what ways, and when differs
- Speculative execution allows a processor to 'guess' about the result of an instruction
  - And either confirm or correct itself later
- A branch predictor bases a guess on the program's previous behavior

Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];
     int x = *address;
     if( x < 100 ){
          y = globalarray[10];
     }
     return y;
}
```

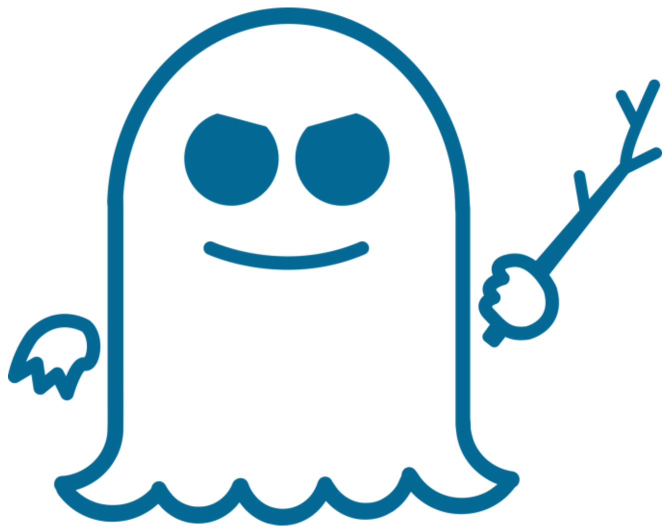
Example: Speculate on *indirect* branch

```
int caller(int(*fptr)()){

    int y = fptr();

    return y;
}
```

```
int foo(){
    return 10;
}

int bar(){
    return 0;
}
```

What happens when we speculate wrong?

- Eventually, a *squash* occurs
  - All work done under the incorrect guess is undone
- Bad guess on branch?
  - Undo everything in the branch!
  - Undo everything related!
- World reverts back to before guess …almost



Example: Speculate on branch

```
int foo(int* address){
     int y = globalarray[0];  // Brought into cache
     int x = *address; // Brought into cache
     if( x < 100 ){
          y = globalarray[10]; // Brought into cache maybe
     }
     return y;
}
```

Speculative attacks

- Three stages:

1. Mistrain predictor

2. Run mistrained code with adversarial input

3. Recover leftover state information

Spectre variant 1

- "Bounds-check bypass"

```
if( x < len(array))
      array[x];
```

Spectre variant 1

- “Bounds-check bypass”

```
if( x < len(array))
      array2[array[x] * 4096];
```

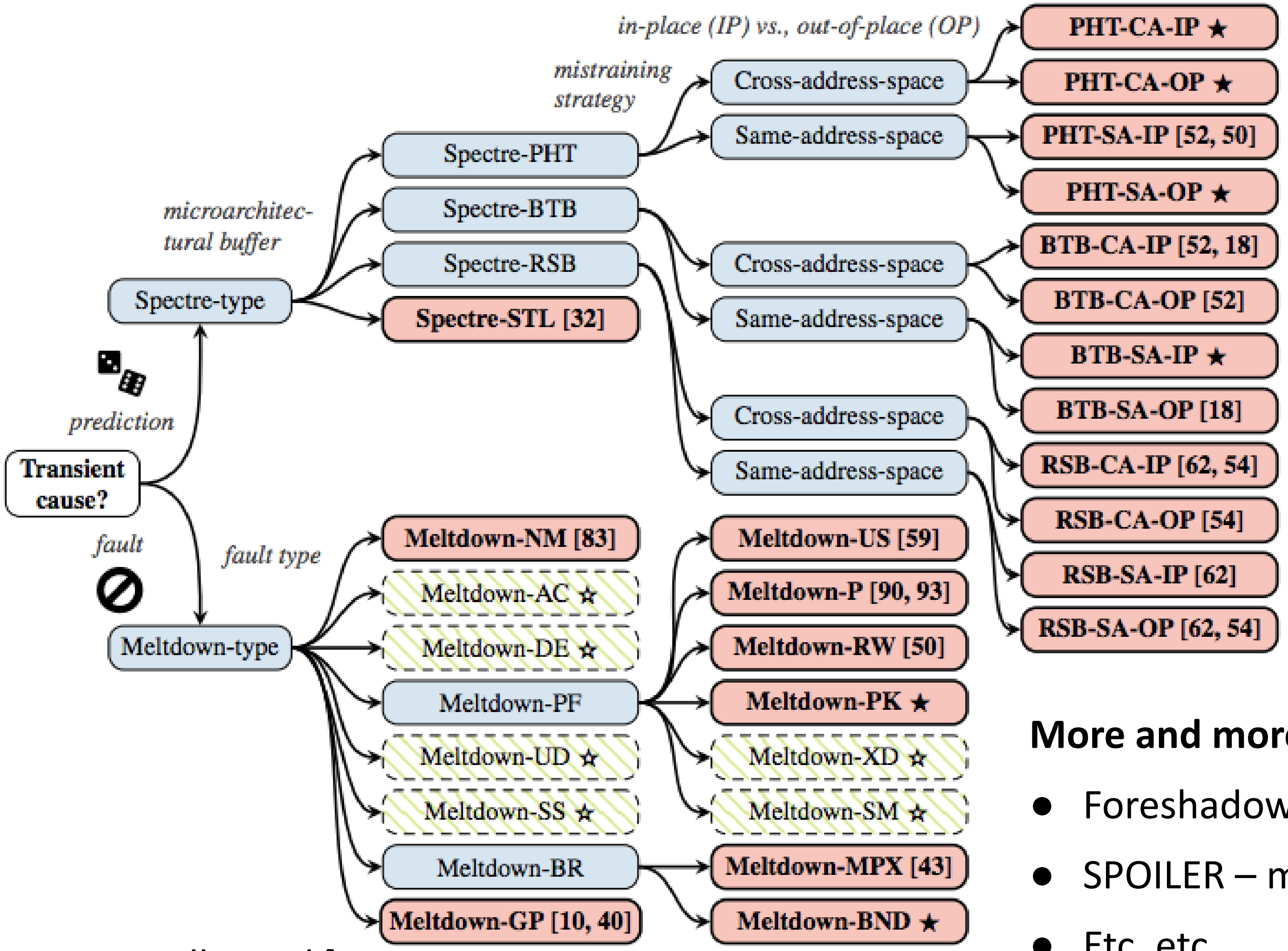
Spectre variant 2

- “Branch target injection”

```
int caller(int(*fptr)()){

     int y = fptr(x);

     return y;
}
```

```
int foo(x){
     array2[array1[x] * 4096];
}

int bar(x){
     return x;
}
```

It’s A Party



More and more:

- Foreshadow – attacks SGX
- SPOILER – mem dependence
- Etc. etc.

[From Canella et al.]

What about ‘Meltdown’?

- Also called Spectre variant 3 (“rogue data cache load”)

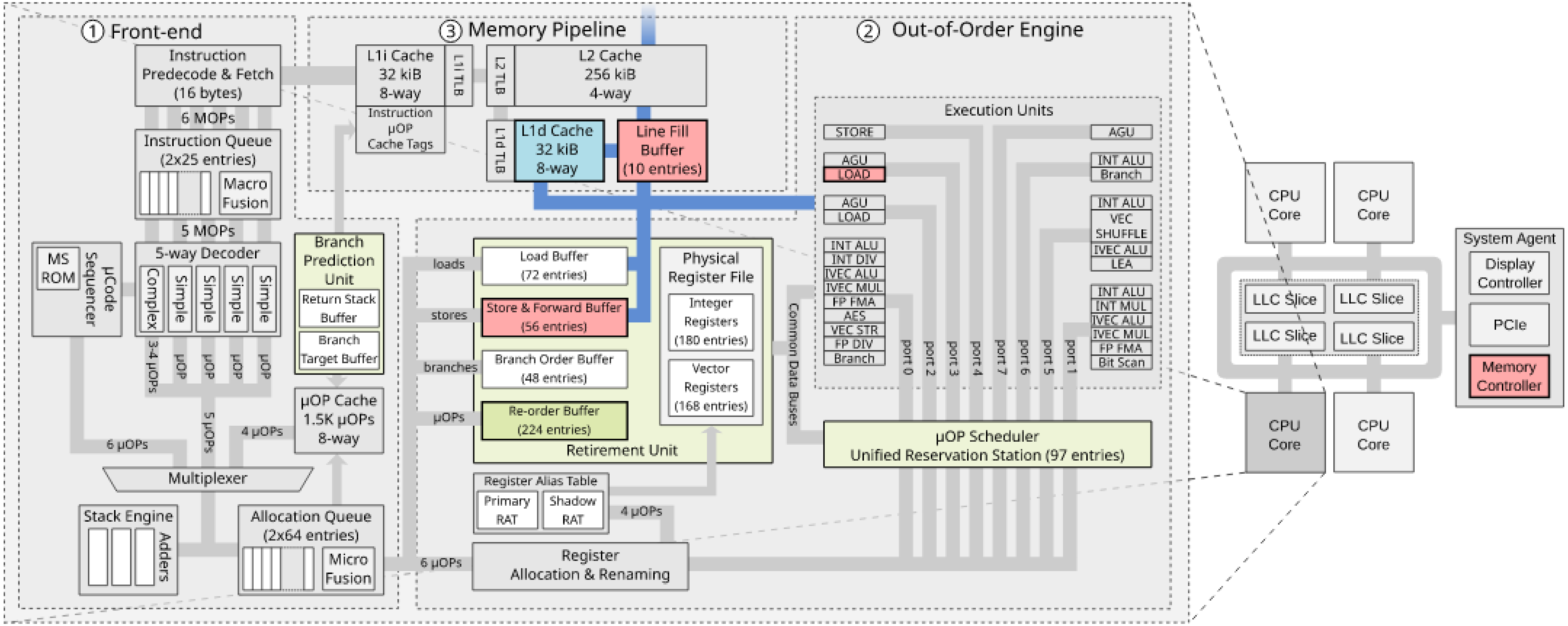
- Spectre v1/v2 require the victim program to have the vulnerable code pattern
  - Just like the victim program has to have a buffer overflow!
  - Spectre is a global problem with speculation conceptually

- Meltdown allows the attacking program to do whatever it wants!

Meltdown: An Intel specific problem

- Memory permissions weren’t checked during speculation
  - At least for some cases

"Imagine the following instruction executed in usermode
```
mov rax,[somekernelmodeaddress]
```
It will cause an interrupt when retired, [...]"



Click on the various components to interact with them. The full interactive version can be found here and the raw SVG can be found here. There is also a more vibrant colored version (the one used in our paper), which can be found here. These diagrams have been made by Stephan van Schaik (@themadstephan).

https://mdsattacks.com/

Canvas

- Browsers had to scramble to deal with Spectre type vulnerabilities as they were exploitable from webpages and allowed for arbitrary memory reads.

- How would you have tried to handle receiving a disclosure like this **as the browser vendors**?
- You can either discuss technical ideas **or** policy objectives for a strategy to handle the vulnerabilities.

Defenses

- Disable User/Kernel memory space sharing
  - KAISER defense
- “Fence” dangerous code patterns
  - Extra instruction that block speculation past some point
- Microcode updates for processors
  - MDS-class fixes

Speculative Attacks wrapup

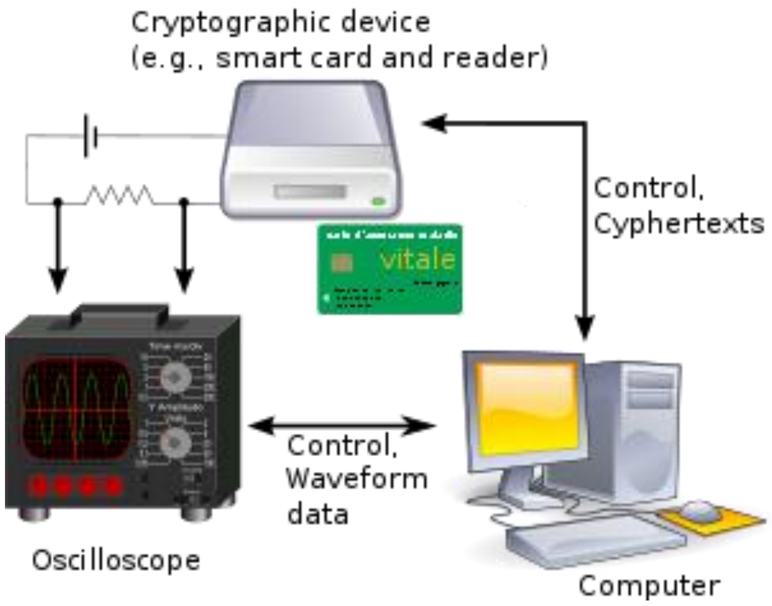
- Spectre vulnerabilities are here to stay, for a long time

- MDS+Meltdown (hopefully) aren't

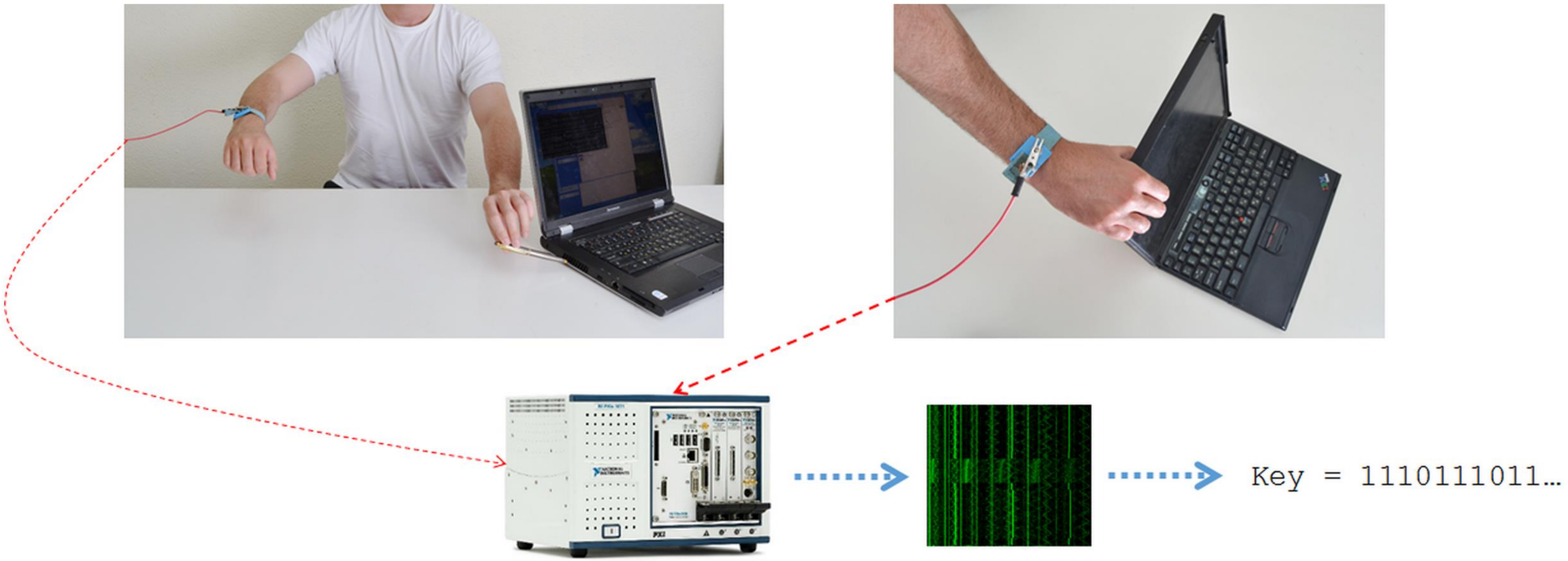
More oddities

Power Analysis

- **Simple power analysis:** Directly read off bits from powerline traces
- **Differential power analysis:** Look for statistical differences in power traces, based on guesses of a key bit



Key Extraction via Electric Potential



Genkin et al. “Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks On PCs” CHES 2014

Identifying Web Pages: Traffic Analysis

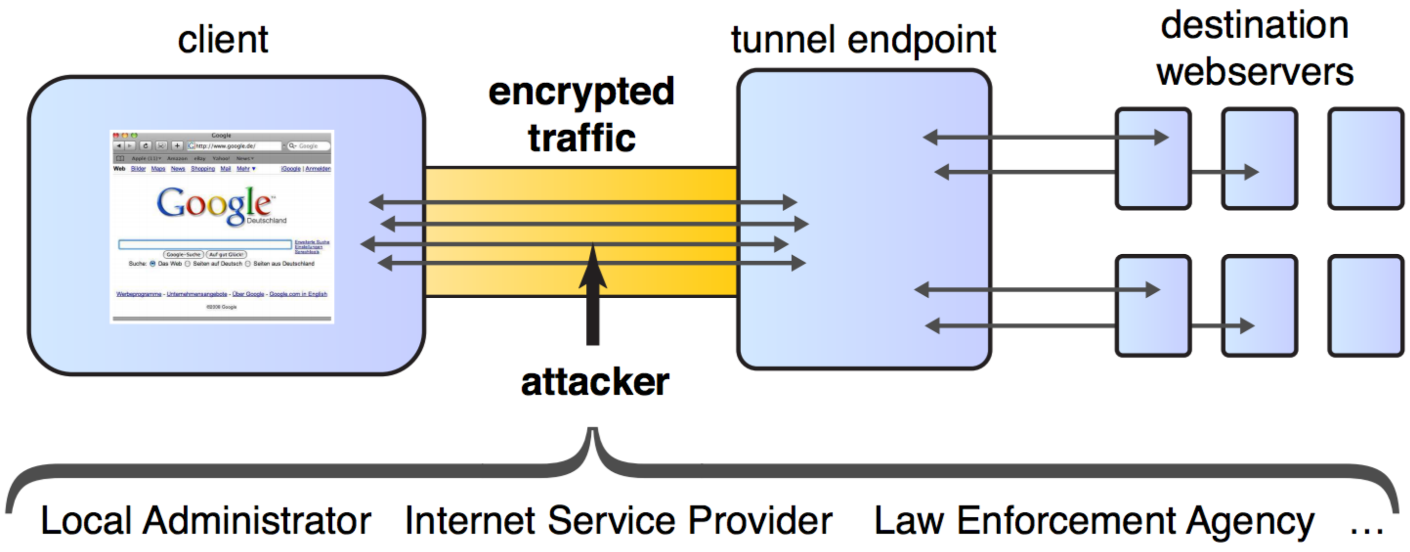


Figure 1: Website fingerprinting scenario and conceivable attackers

Herrmann et al. "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier" CCSW 2009

Powerline Eavesdropping

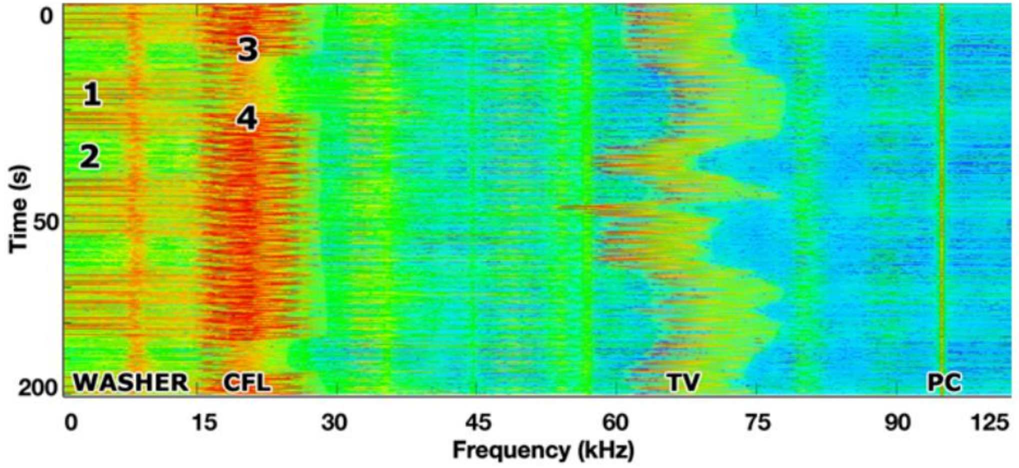


Figure 1: Frequency spectrogram showing various electrical appliances in the home. Washer cycle on (1) and off (2). CFL lamp turning off briefly (3) and then on (4). Note that the TV's (Sharp 42" LCD) EMI shifts in frequency, which happens as screen content changes.

Enev et al.: Televisions, Video Privacy, and Powerline Electromagnetic Interference, CCS 2011

Identifying Web Pages: Electrical Outlets

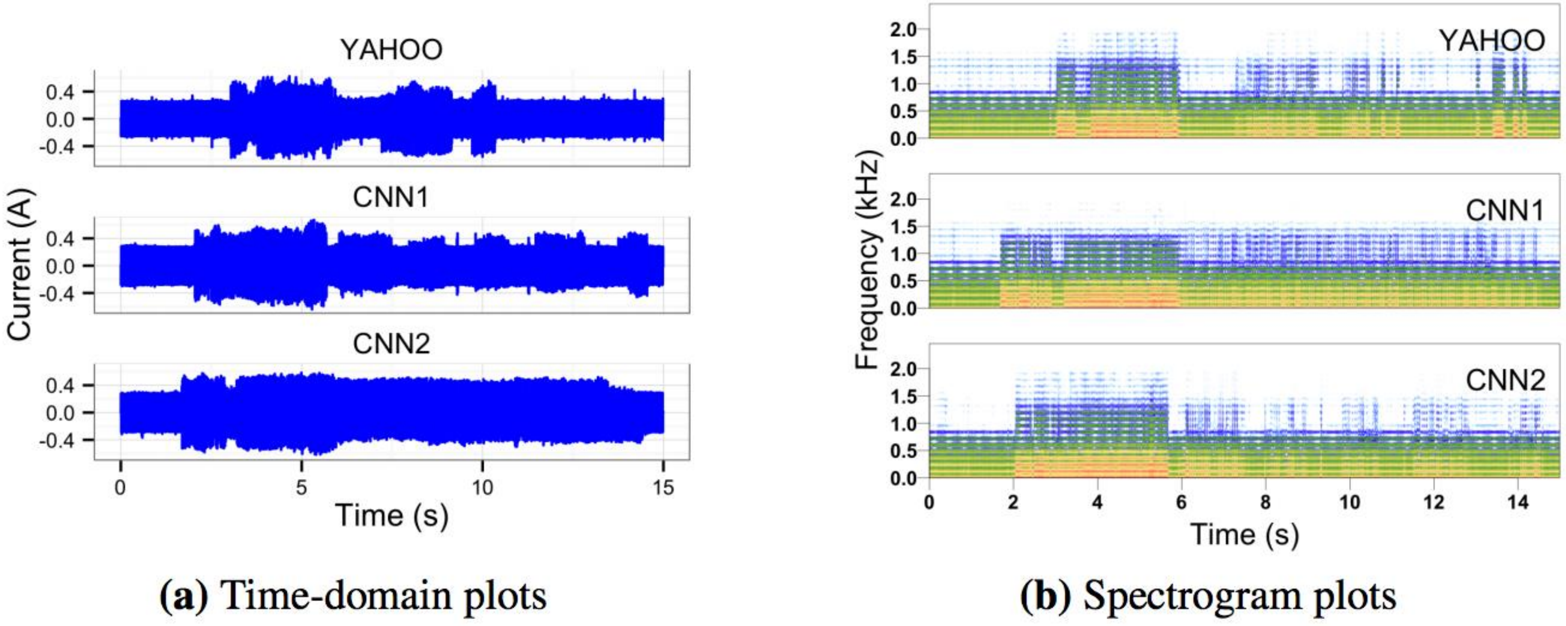


Fig. 1: Time- and frequency-domain plots of several power traces as a MacBook loads two different pages. In the frequency domain, brighter colors represent more energy at a given frequency. Despite the lack of obviously characteristic information in the time domain, the classifier correctly identifies all of the above traces.

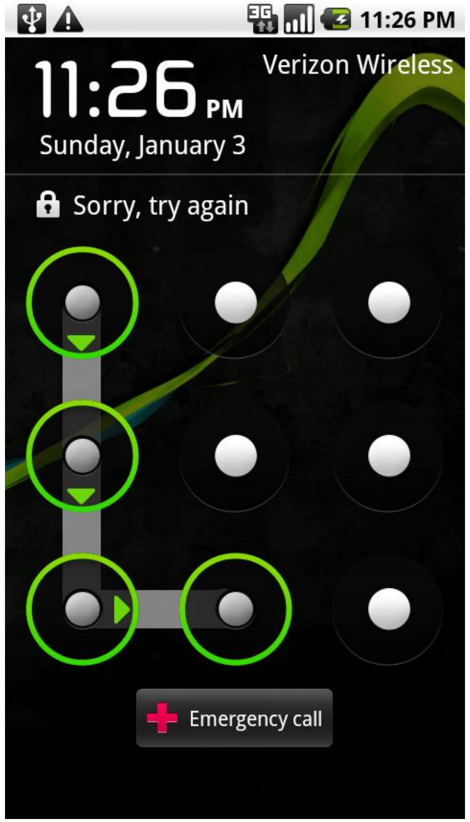
Clark et al. "Current Events: Identifying Webpages by Tapping the Electrical Outlet" ESORICS 2013

More Examples…

Keyboard Eavesdropping

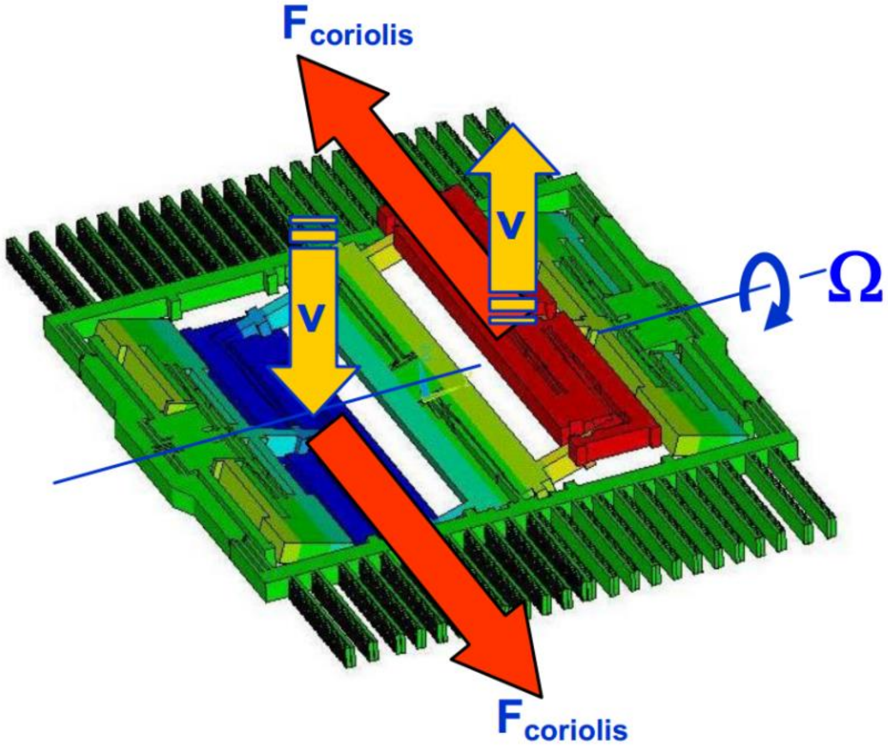
Zhuang et al. “Keyboard Acoustic Emanations Revisited” CCS 2005
Vuagnoux et al. “Compromising Electromagnetic Emanations of Wired and Wireless Keyboards” USENIX Security 2009

Accelerometer Eavesdropping



Aviv et al. "Practicality of Accelerometer Side Channels on Smartphones" ACSAC 2012

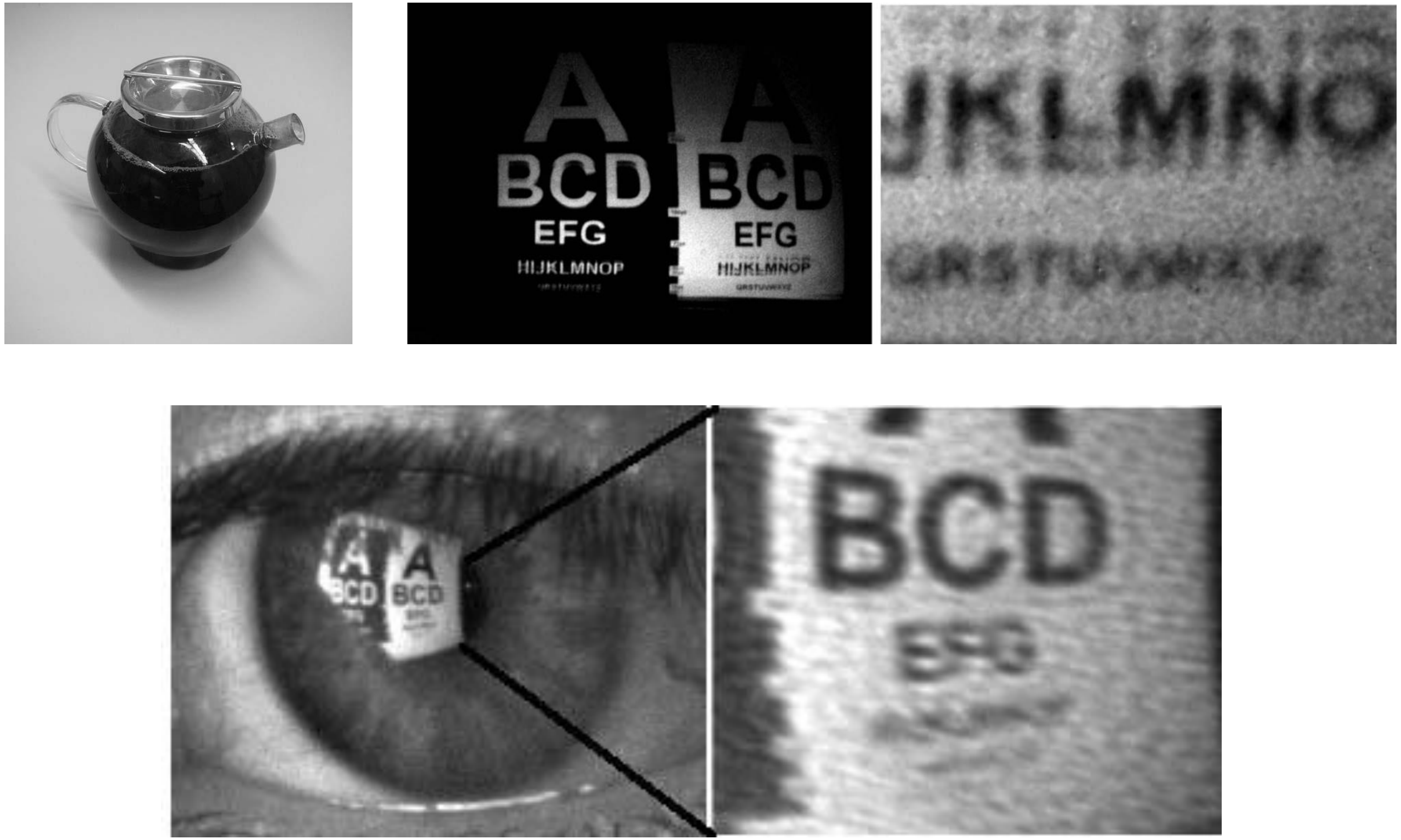
Gyroscope Eavesdropping



Michalevsky et al. “Gyrophone: Recognizing Speech from Gyroscope Signals” USENIX Security 2014

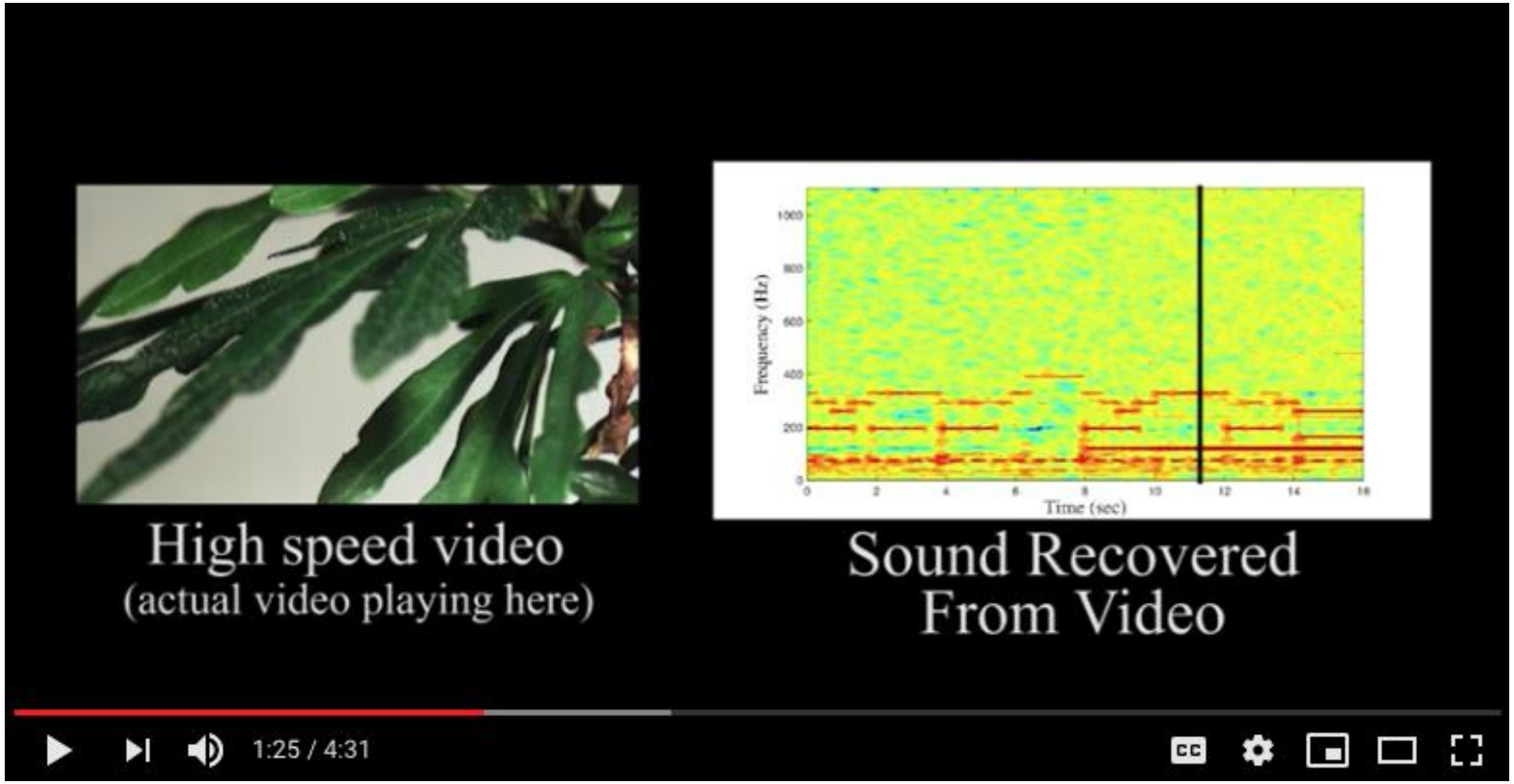
[Backes et al.]

Compromising Reflections



Audio from Video

https://www.youtube.com/watch?v=FKXOucXB4a8



Davis et al. “The Visual Microphone: Passive Recovery of Sound from Video” SIGGRAPH 2014