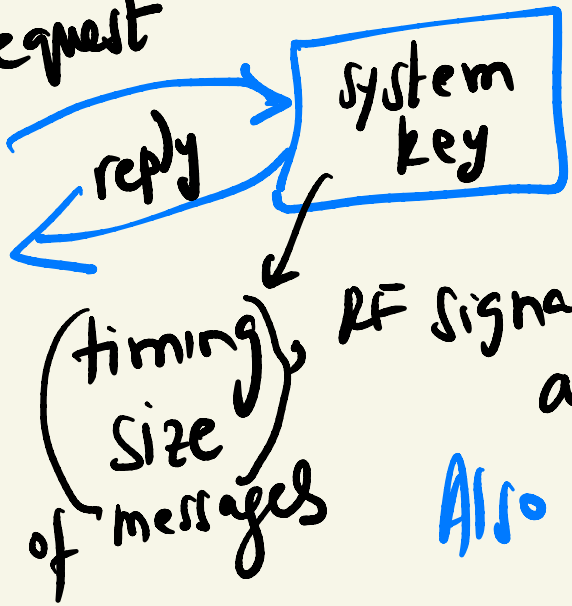
CPU Timing Attacks

Side channels: reveal info in an "unintended" way



Also called metadata

Cache affected by req/reply at time t, affects req/reply at time t + T.

Examples

TEMPEST: screen showing info (CRT screen). Electron beam produced EM emissions.

Linux dm-crypt: encrypt contents of disk on the fly. CPU cache timing attack: extract kernel key [Osvik et al. 2006]

Smart cards: power analysis CPU runs code depending on the key. "0" and "1" use different ops & power.

HertzBleed (2022)

Browser page cache: use timing in Javascript to decide if user has visited a page or not.

Spectre

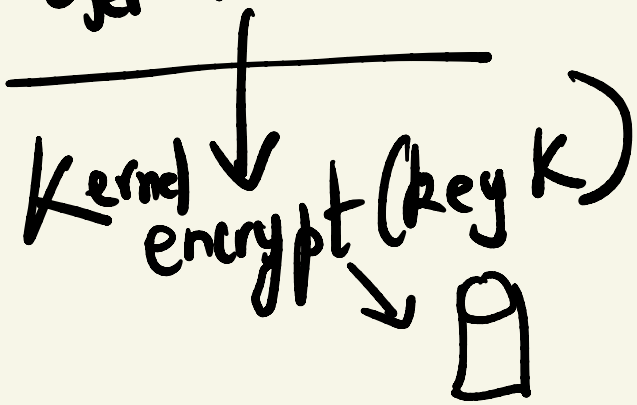
complicated, will build up to it.

- cache timing (dm-crypt)
- speculative execution (melt down)
- branch prediction (Spectre)

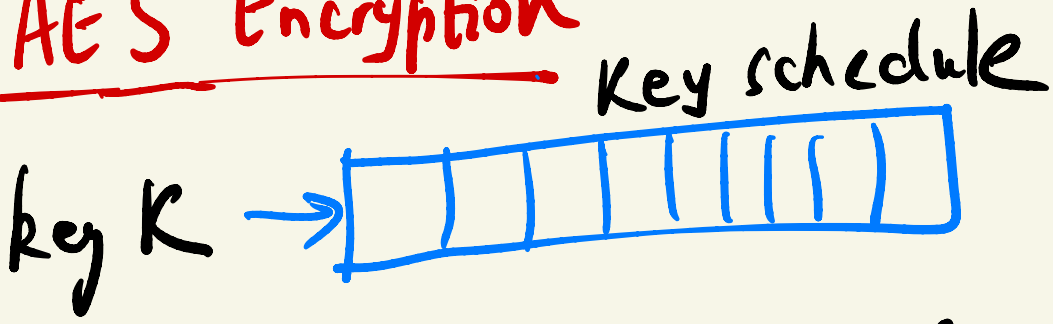
prg. control flow ← faulting instr.

Linux dm-crypt

User write



AES Encryption



data → encrypt ↗ b XOR kschedule^{i} = b'

b' XOR kschedule^{j} = b''

...

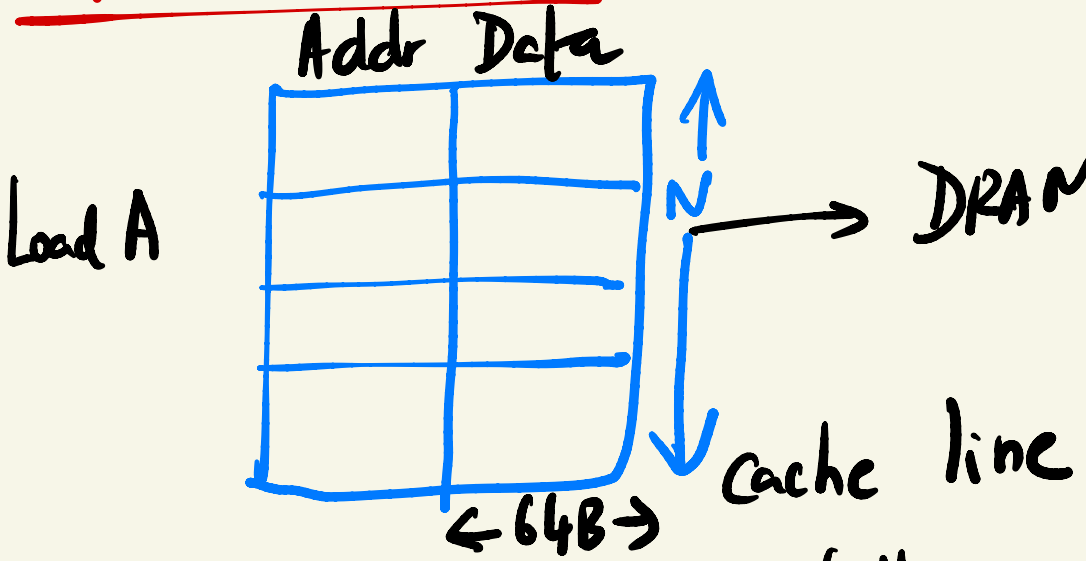
(1) Assume adversary knows b

(2) Adversary sees memory accesses
i.e., i, j

i, j depend on K and b, b', ...

figure this out!

CPU Caches



A: any place in cache, fully associative: expensive!

"hash" A: A % N is where it is going to be in cache.

Cache timing attacks exploit collisions

only 64B can be stored in a row (in general a multiple of 64B)

AES Timing Attack.

Adv

(1) Fill the cache

(2) Issue a write to Kernel for data d

Kernel

runs encryption on d and brings some data into cache from keyschedule

(3) Access the same memory locations as in (1). Record how long they take. If fast, was it evicted by kernel. If slow, was evicted. Figure out address of data that kernel accessed.

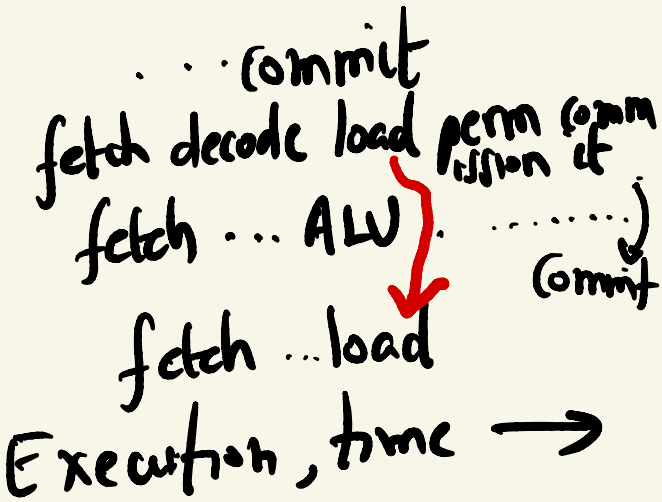
noisy same → repeat

info about i,j

Speculative Execution

```
add r1, r1, r2
mov [a] → r1
add r2, r3, r4
mov [r1] → r3
```

can take 100's of cycle



Commits always in order

1. mis-speculation → undo r4
2. CPU forwards values between instructions → in pic. above

Meltdown

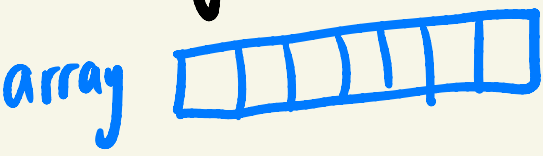
Speculative protection checks

User

mov [a], r1
mov array[r1], r2

Ker

key ← address a



fetch load .. perm .. commit
fetch load perm ..

will commit, but next instr is kernel exception handler, not mov
array[r1] modifies cache contents!

side effects not undone

Back to Cache Timing Attack

→ Fill cache

```
[ mov [a], r1
  mov array[r1], r2
```

→ Check what's in cache.

Can scan kernel memory. Kernel maps all of physical memory to its space → know this. Access well known memory locations, eg: screen.

→ don't see `array[r1]` but know part of `r1` value, which is value at address `a`

Secret-dependent memory accesses are a no-no.

Mitigations for M.D.

Hardware: don't do perm check after load, don't forward values

Software: Linux has 2 page tables, one for user processes and some kernel pages, other for kernel. syscall → change the page table

Branch Prediction

```
cmp r1, r2
beq foo
add r1, r2, r3
foo: sub ...
```

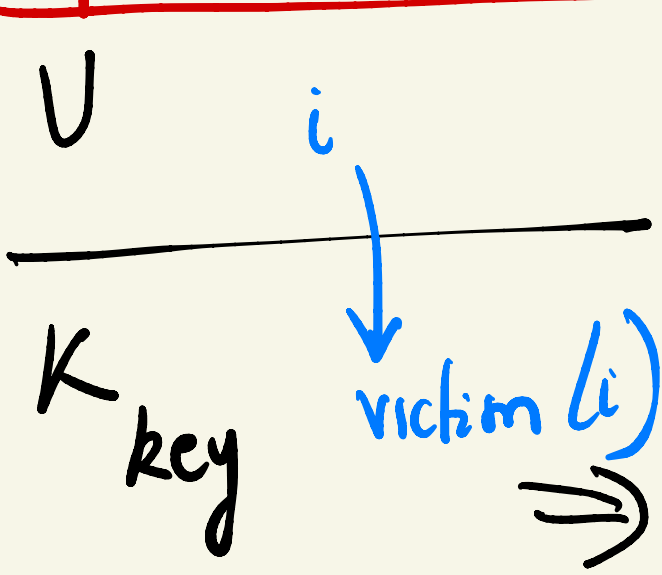
fetch ... cmp

??

f(PC of beq)

need to abort on misprediction

Spectre Attack



a) attacker wants to go inside the if when $i \geq sz$!

b) Make sure sz is not cached.

c) Massage branch predictor to predict $i < sz$

d) array2[v] Loaded into cache

getting info on v

Mitigations for Spectre

Software

C1: reduce accuracy of timers in JavaScript by adding jitter.

C2: Use serializing instructions like lfence on both outcomes of a branch.

C3: WebKit replaces array bound checks with index masking (> lagerval)

Linux: array [i % N]

software change