Lecture 18: Hardware security

6.1600 - Fall 2023

Corrigan-Gibbs & Zeldovich
MIT

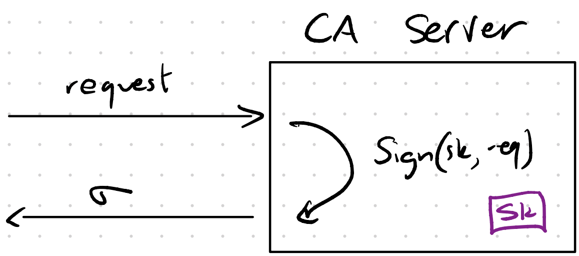
Hardware Security

- Randomness Failures
- Attacks w/o physical access
  * Rowhammer
  * Timing attacks
  * Cache attacks
  * Spectre
- Attacks w/ physical access
  * Probing attacks
  * Fault attacks
  * Supply-chain attacks

Logistics

Running example...

* You are running a CA that signs certs
  (Or: think of a cryptocurrency exchange)
* Business is premised on keeping secret key secret.
  ↳ Juicy target - small $ secret



Should enforce some policy...

only sign certs for *.mit.edu domains.

→ Can prove security of signature scheme under crypto assumption.

→ Can verify that crypto implementation faithfully implements sig algorithm... (on some ideal h/w)

... then you buy a computer, load the code onto it and run it on a machine with a bunch of other software

↳ What can go wrong?
↳ How to protect?

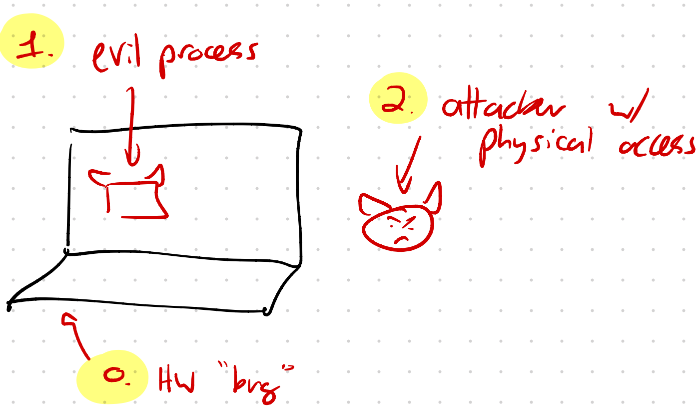
Theme:

* We typically use (implicitly) very simplistic model of hardware
  - e.g. our crypto defns (CCA, etc) say nothing about how long different operations take and whether this could leak info

* The gap between model & real HW is the source of many bugs & attacks

⇒ Not often clean/satisfying solutions.

Three types of attack we'll discuss...



Randomness failure : Random bits not random

* All of the crypto schemes we've discussed require randomness: secret keys, nonces (CPA), ...

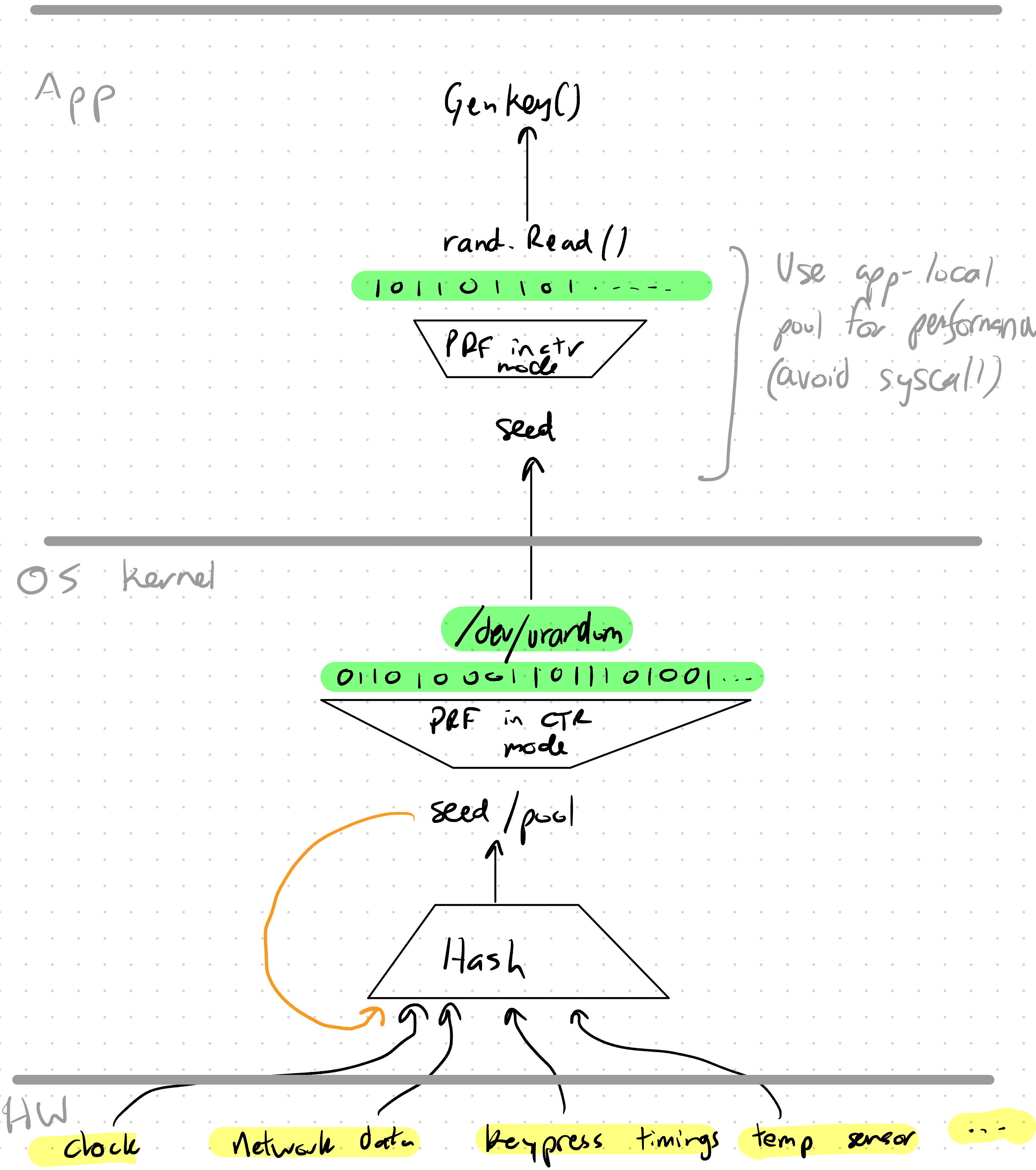
* You've seen in labs how bad randomness can cause ECDSA secret to leak.
  - ↳ Problems go far beyond that (Encryption, auth, etc)

* Randomness bugs are common
  - ↳ Not clear how to test for them.
  - ↳ Buggy software seems to work fine.

Examples from THIS YEAR

CVE-2023-1732 - Sig alg doesn't check rand error code
-42820 - Seed for PRG leaked (key)
-36093 - Password reset token
-31147 : uses rand() instead of CS PRG

How randomness works in PC



Randomness failures

HW: * Embedded devices gen SSH keys at first boot
* HW rand sources aren't great at boot
↳ many generated weak/guessable keys

IDEA: RDRAND instruction on CPU

OS: * Bugs not as common

App: * Use time() or other non-random src as seed
* Use weak PRF to stretch seed from OS
BAD: rand() in C, random.randint in Python
↖ only use for sci computing (non-sec apps
* VM duplication or fork()
↳ Both app clones have same RNG state

How to fix???

Attacks

without

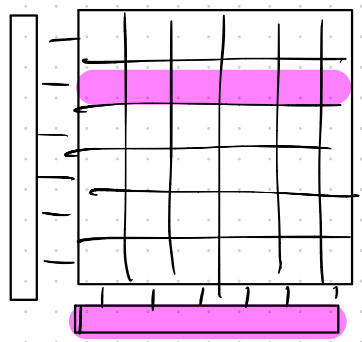
physical

access...

Rowhammer

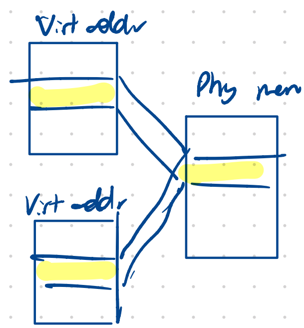
Surprise: By reading memory often, can induce bit flips in adjacent memory locations.

- Data in main mem (DRAM) stored in capacitors
  - ↳ They drain over time, must be "refreshed" (64 ms)
- Reading chunk of mem drains capacitors
  - ↳ must be rewritten
- Voltage fluctuations on one row cause neighboring rows to discharge more quickly



Attack: Read bytes in mem as fast as possible
⇒ Bit flips in nearby memory.

OS deduplicates memory pages
↳ If attacker process & victim process have chunk of identical data, OS stores them both in same phys RAM



⇒ Attacker can hammer arbitrary RAM locations!

Mitigation: Refresh potential victim rows more often
↳ HW changes.

Timing Attack: Example

DSA signatures compute $g^r \bmod p$ for secret $r \approx 2^{2048}$

↳ If attacker learns "r" it can recover the secret signing key. $(p, g \approx 2^{2048})$

The expensive bignum mul happens once per "1" in the secret value r.

↳ Time leaks Hamming weight (# of ones) of exponent

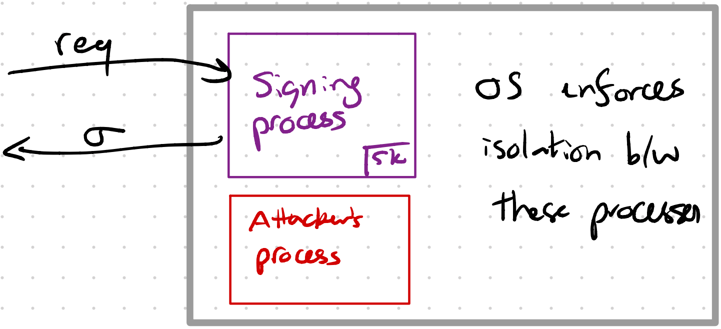
Fix? Constant-time code where possible.

```
T = { g, g^2, g^4, g^8, g^16, ....., g^(2^2048) }
r = r1 r2 r3 .... r2048
out = 0
for i = 1, ..., 2048 {
    if ri = 1 {
        out *= T[i]  (mod p)
    }
}
return out
```

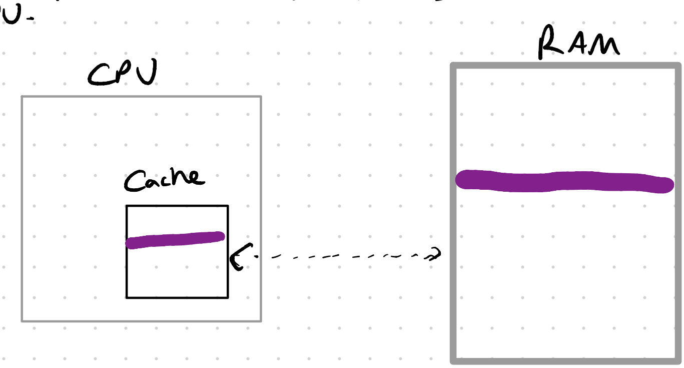
Cache Attacks (or: shared HW resources more generally)

Ex from Yarom & Benger 2014... simplified

* Memory accesses take diff amts of time depending on history of accesses



Problem: * Attacker & victim process run on same CPU.
* Victim can leave traces of secrets in state of the CPU.



1. Victim runs, loads purple line or not
2. Attacker runs, loads purple line.
3. If access is fast, attacker knows that victim loaded purple.

↳ Attacker learns info about victim's access pattern.

Cache Attack: Example

DSA signatures compute $g^r \bmod p$ for secret $r \approx 2^{2048}$

↳ If attacker learns "r" it can recover the secret signing key. $(p, g \approx 2^{2048})$

If attacker & victim both use same crypto library, the OS will keep only one copy of library in phys RAM.

1. Victim runs
2. OS interrupts, runs attacker
3. Attacker runs, tries to access "A"
4. If fast, then $r_i = 1$

... repeat to get all bits of r.

```
T = { g, g^2, g^4, g^8, g^16, ...., g^(2^2048) }
blah = 0
out = 0
for i = 1, ..., 2048 {
    if r_i = 1 {
        out *= T[i]      <- A
    } else {
        blah *= T[i]     <- B
    }
}
return out
```

Cache attacks: Defenses?

Problem: * Hard to specify limits on leakage b/w processes via "microarchitectural side channels"

* CPU vendor keeps proc internals secret

Only "complete" answer: Use separate hardware for security-critical code.

No sharing of cache → No cache attacks

... still need to be careful about timing, etc.

Examples

* Hardware security modules
* U2F token
* co-processors for crypto (next time)

Spectre: Speculative-Execution Attacks

Surprise: CPUs execute branches "not taken"
↳ can leak secrets... many many variants

* Reading from RAM is relatively costly

Idea:
- Guess whether "if" condition is T/F
- Execute branch before getting answer from memory
- If guessed wrong, rerun correct branch.
  ↳ Problem: CPU doesn't completely reset its state when rerunning

Example
- attacker provides x
- array1[x] is secret value

```
if (x < len(array1)) {
    y = array2[array1[x] * 4096];
}
```

} From Spectre paper IEEE S&P 2019

1. attacker provides x too large
2. CPU mispredicts branch, executes next line
3. CPU evaluates branch, cleans up

array2[secret] is cached!
↳ Use cache attack to extract

Spectre

- Many many variants...
  ↳ Speculation tampers w/ many parts of CPU
    see https://transient.fail

Mitigations

* No single solution (since not a single attack)
* Process isolation... useful e.g. in browser
  ↳ Data doesn't leak across process boundaries
* Some hacks ("retpolines") help ???
  ↳ Replace indir branch with returns... try to prevent speculation past indir branch

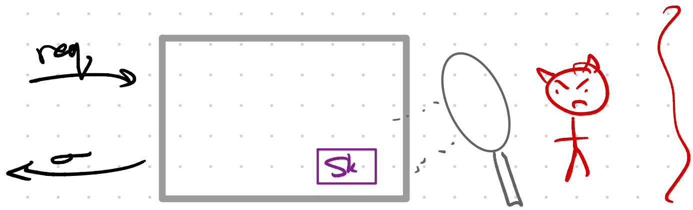
Now, on to:

Physical Attacks

Physical side-channel leakage

Another threat: Internal state of HW can leak to attacker → Can defeat "airgap"

e.g. CA signs cert, leaks private key in the process



Examples:

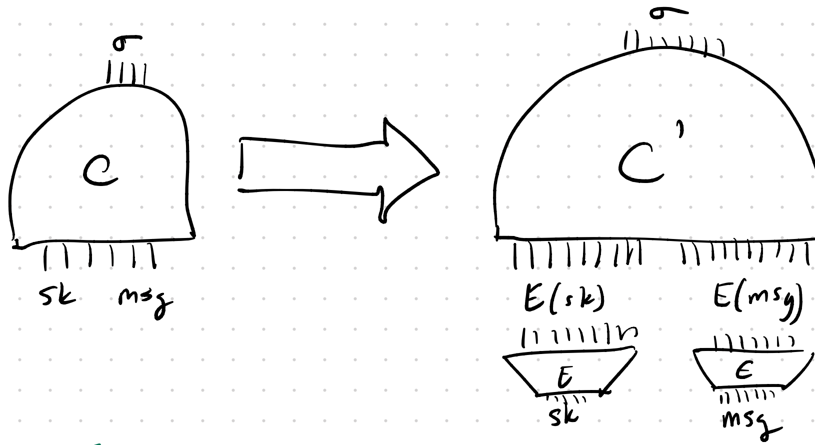
- power analysis — power use depends on secret
- probe on pins of chip
- EM emissions (TEMPEST attacks)
- blinking lights on network router/HDD light
- Hertzbleed - clock speed depens on secret
- audio

Even 0.001 bits of leakage/sec is enough to leak a secret key in a few hrs.

Probing Attacks: Defense

Assume: Attacker can only probe values on t internal wires of signing circuit.
↳ Intuition: Probes are $$$

Then, there's a clever defense against probing attacks: "masking"

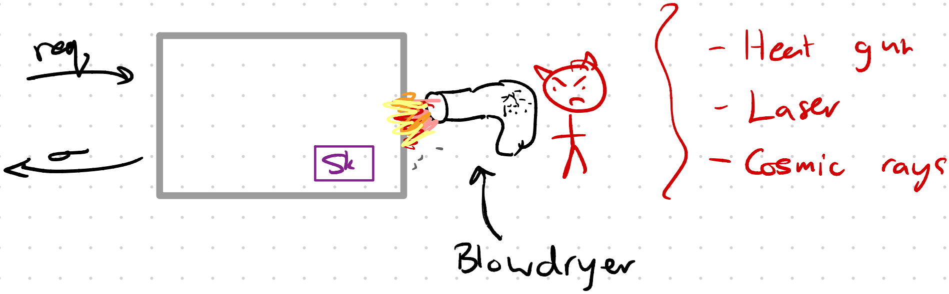


IDEA: Take a boolean ckt C implementing sig scheme.
Convert to ckt C' that implements sig scheme.
BUT — looking at any t internal wires of C' "leaks nothing" about sk.
(Technique: Again secure multiparty computation)

Still, only a very partial solution....
- input wires leaky
- what if attacker can get values on t+1 wires?

Fault Attacks

Another threat: Attacker induces "faults" (bit flips) in computation.



One bit flip can cause chaos:
- Leak secret signing key (RSA)
- Corrupt kernel data structures
  ↳ Attacker can hijack machine

* If we assume that adv can't flip "too many" bits, can defend similar to probing attacks.
  ↳ Replicated HW used e.g. on satellites

* No great soln: For RSA sigs, verify sig before outputting (very special case)

Supply-Chain Attacks

- Attacker modifies the computer on its way to you
  * Snowden slides — Cisco router ... unlikely?
  * Hardware wallets on eBay ... likely?

- Modifications
  - mgmt interface
  - randomness
  - preloaded keys
  - extra comm

- How to defend? No great solns...
  * Buy in cash at random store? Doesn't really work for HW wallets
  * Inspect? Easy to hide... e.g. randomness, transistor
  * Build yourself? ☹
  * Trustworthy suppliers? DoD

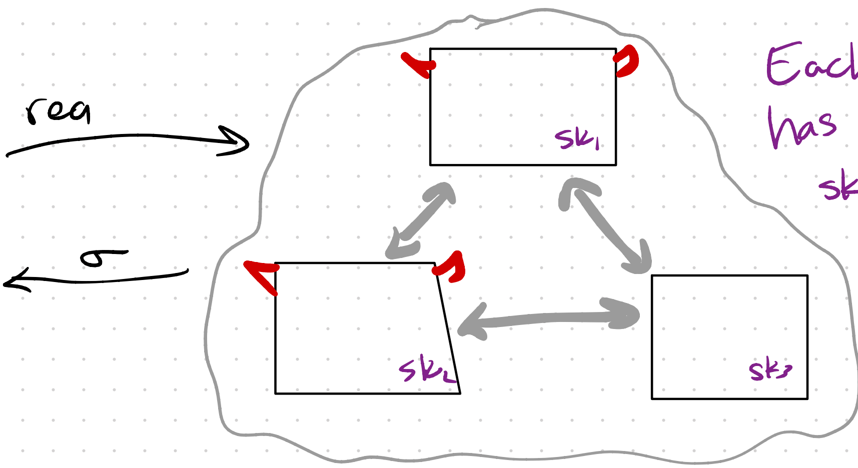
↳ If the hardware is adversarial, you don't have a secure foundation

Supply-Chain Attacks

One meaningful defense: thresholding / "splitting trust"

Idea: Build system out of N computers, assume that attacker can compromise at most N-1 of them.
↳ Precise limit on attacker's power.

e.g. signing service w/ policy enforcement (1 BTC/day)



Each device has a "piece" of sk
$sk = sk_1 \oplus sk_2 \oplus sk_3$

- As long as attacker doesn't compromise all signing servers, can't learn sk. or violate policy.

  "Secure multiparty computation"

- Possible in theory to distribute any computation
- In practice, works only for simple comps