# CSE 484/M584: Computer Security (and Privacy)

Spring 2025

David Kohlbrenner
dkohlbre@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner, Nirvan Tyagi. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

# Admin

- Lab 4 : you should have started looking at tinyserv.
  - If you haven’t start today.
  - I only see 34ish forks!

# tinyserv – a tiny, bad, HTTP server

- ~1000 lines of C code
- Moderately well commented
- Quite buggy
- You can interact with it via command line tools or a web browser

# tinyserv – a tiny, bad, HTTP server

- ~1000 lines of C code
- Moderately well commented
- Quite buggy
- You can interact with it via command line tools or a web browser
- 4 exploits
  - Exploit 1: Given RCA, create patch (Part A)
  - Exploit 2: TAs covered in section!
  - Exploit 3 & 4: Given exploit, create RCA (Part B), create patch (Part C)

# Lab 4 Components

- Part A
  - We will give you the RCA for an exploit, and you have to write the patch
- Part B
  - We will give you an exploit, and you have to write the RCA
  - (You can choose one of two exploits. You may do the other for extra credit.)
- Part C
  - You have to write the patch for Part B's exploit
  - Do not wait for feedback from Part B! We will try to get it to you before Part C is due but cannot guarantee it.

# Major Features

- “admin” login
  - Sets a randomized password on server start
  - Successful login sets a cookie that lets admins access admin content
  - admin.txt contains a log of requests received so far
  - /admin_only/index.html secret admin homepage
  - (Our exploits work by demonstrating they can access admin.txt)
- Dynamic content fills
  - Some pages have dynamic content (notably 404s) that gets filled at request
- Response caching
  - Pages are cached in a hashtable on first send
  - Future responses will check the hashtable first

# RCA Strategies

- Read through the server code (see tinyserv.c to start)
  - You don’t have to understand everything!
- Read through the exploit inputs and try to guess which parts of the tinyserv code might be related; start debugging there!
- Use gdb for debugging and execution tracing
  - gdb --args ./tinyserv ./files
  - break [function name or line number]
  - run
  - From another terminal window, you can now run the exploits
- (Maybe:) Modify tinyserv.c to test things or add print statements
  - You may want to add your own logging to the server.

# Patching Reminders

- Try a variety of workflows to make sure you didn't break anything
- At least anything *new*
  - Eg tinyserv will exit if you try to request a directory. You shouldn't be fixing that.
- Try basic things in an *unpatched* tinyserv, and your *patched* tinyserv:
  - Visit a variety of the pages
  - Log in and log out
  - Restart the server
  - Log the status of caching
  - Etc.

# Patching writeup reminders

- This is mandatory, but also a chance to explain your changes.

- There are common patching strategies that *will change* some behavior of tinyserv.
  - If you document and justify that change, (usually) no point deduction.
  - If you don’t document that change, some point deduction.

# Randomness and Entropy

We've been using it, but how should we be using it?

# Ingredient: Randomness

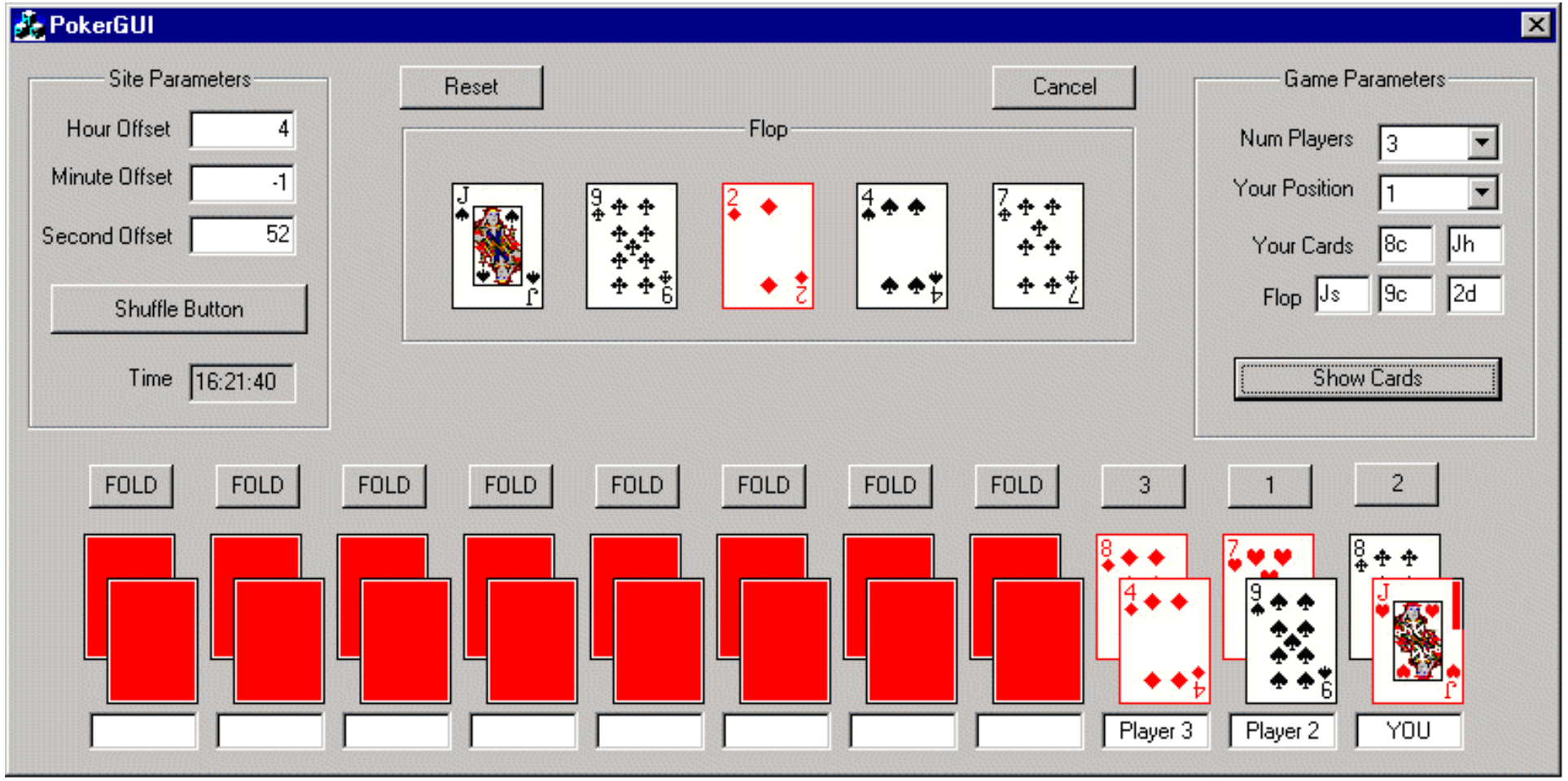
- Many applications (especially security ones) require randomness
- Explicit uses:
  - Generate secret cryptographic keys
  - Generate random initialization vectors for encryption
- Other “non-obvious” uses:
  - Generate passwords for new users
  - Shuffle the order of votes (in an electronic voting machine)
  - Shuffle cards (for an online gambling site)

# C's rand() Function

- C has a built-in random function: `rand()`
- Don't use `rand()` for security-critical applications!
  - Given a few sample outputs, you can predict subsequent ones

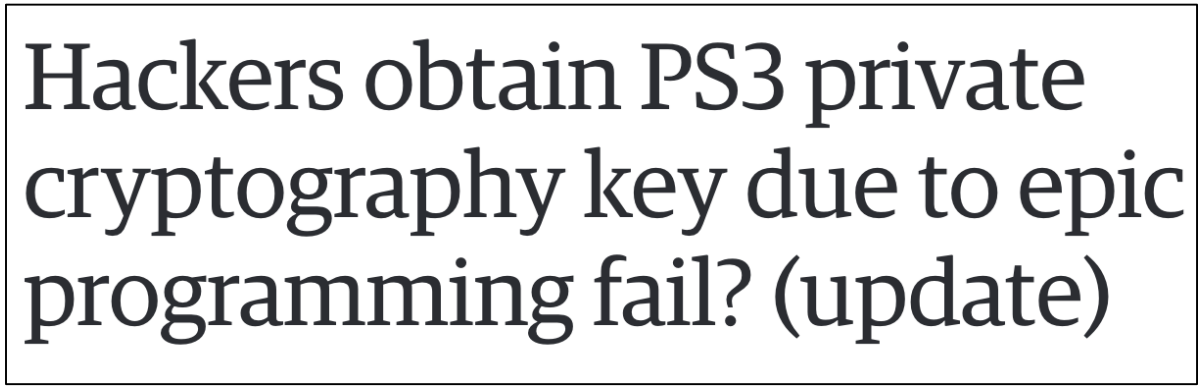
```
unsigned long int next = 1;


/* rand:return pseudo-random integer in 0-32767 */
int rand(void) {
        next = next * 1103515245 + 12345;
        return (unsigned int)(next/65536) % 32768;
}


/* srand:set seed for rand() */
void srand(unsigned int seed) {
        next = seed;
}
```


A World of Action!
Help
Suggestions
Fold in Turn
$0
jonnyboy
brute
mamajoe
RickG
Fold
Check
Bet ($5)
Options
Sit Out
Leave
584330
mamajoe: Hey guys, Big B is in!
Dealer Text




More details: “How We Learned to Cheat at Online Poker: A Study in Software Security”
https://web.archive.org/web/20120301022831/http://www.cigital.com/papers/download/developer_gambling.php

# PS3 and Randomness




http://www.engadget.com/2010/12/29/hackers-obtain-ps3-private-cryptography-key-due-to-epic-programm/

- 2010/2011: Hackers found/released private root key for Sony's PS3
- Key used to sign software – now can load any software on PS3 and it will execute as "trusted"
- Due to bad random number: same "random" value used to sign all system updates

# A recent example: keypair

https://securitylab.github.com/advisories/GHSL-2021-1012-keypair/

- keypair is a JS library for generating (asymmetric) keypairs

The output from the Lehmer LCG is encoded incorrectly. The specific line with the flaw is:

```
b.putByte(String.fromCharCode(next & 0xFF))
```

The definition of putByte is

```
[…]putByte = function(b) { this.data += String.fromCharCode(b); };
```

Since we are masking with 0xFF, we can determine that 97% of the output from the LCG are converted to zeros. The only outputs that result in meaningful values are outputs 48 through 57, inclusive.

The impact is that each byte in the RNG seed has a 97% chance of being 0 due to incorrect conversion. When it is not, the bytes are 0 through 9.

# How might we get “good” random numbers?

- Gradescope!
- Where can we get (as in, where in the world) *random* bits?
- Do random events actually exist?

# Obtaining Numbers

- For security applications, want "cryptographically secure pseudorandom numbers"

# Obtaining Pseudorandom Numbers

- For security applications, want "cryptographically secure pseudorandom numbers".

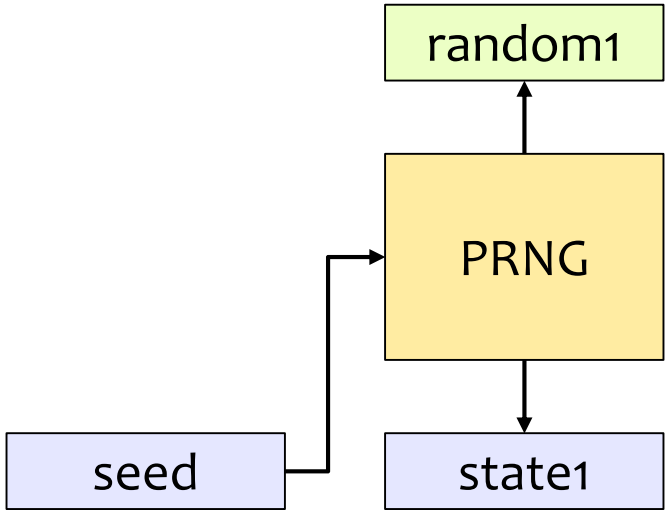
- Numbers that cannot be predicted by a bounded adversary.

# Obtaining Pseudorandom Numbers

- For security applications, want "cryptographically secure pseudorandom numbers".
- Numbers that cannot be predicted by a bounded adversary.
- Side-step the entire philosophical question!

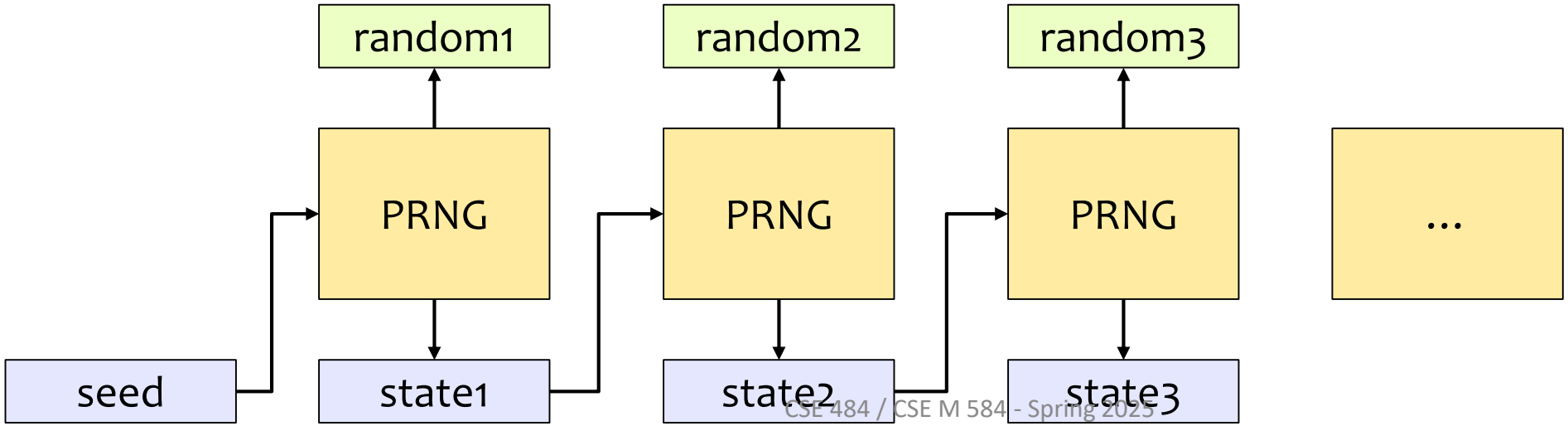
# Obtaining Pseudorandom Numbers

- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)



# Obtaining Pseudorandom Numbers

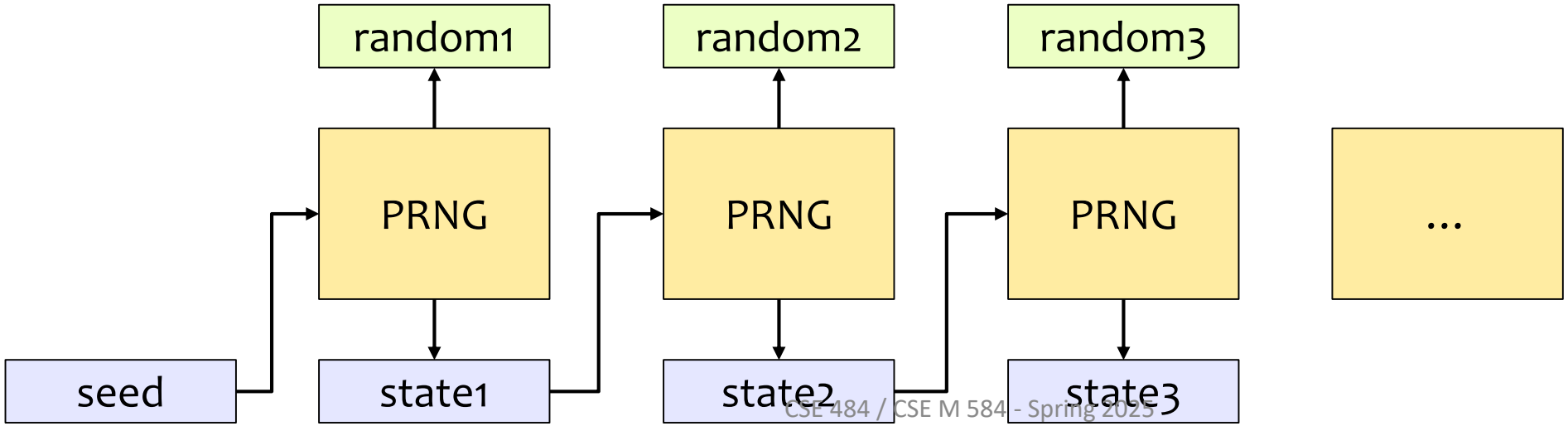
- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)



# Obtaining Pseudorandom Numbers

- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)

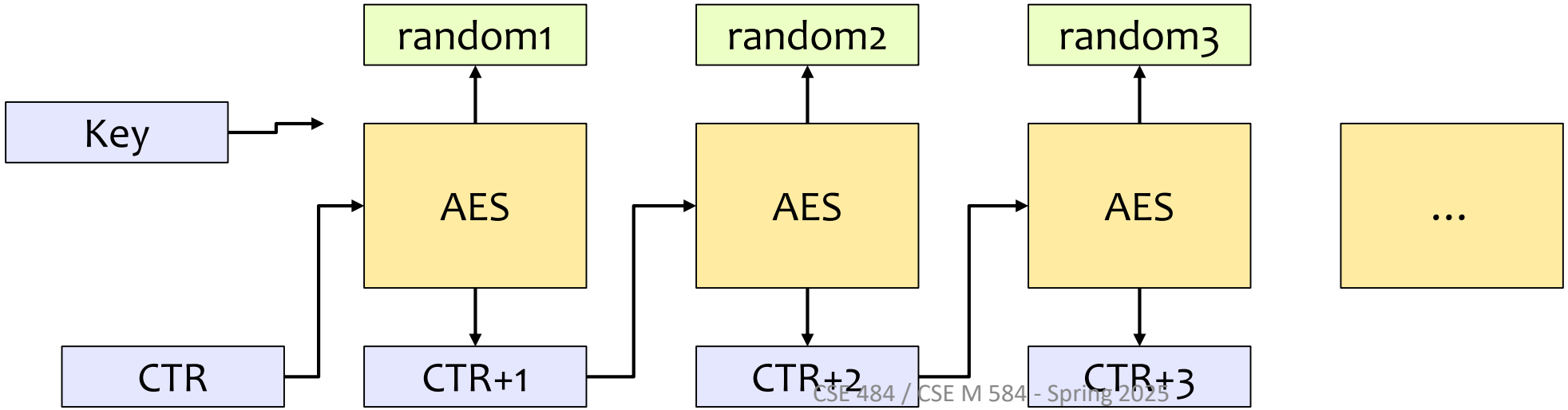
- We've seen something that works like this for a computationally-bounded attacker!



# Obtaining Pseudorandom Numbers

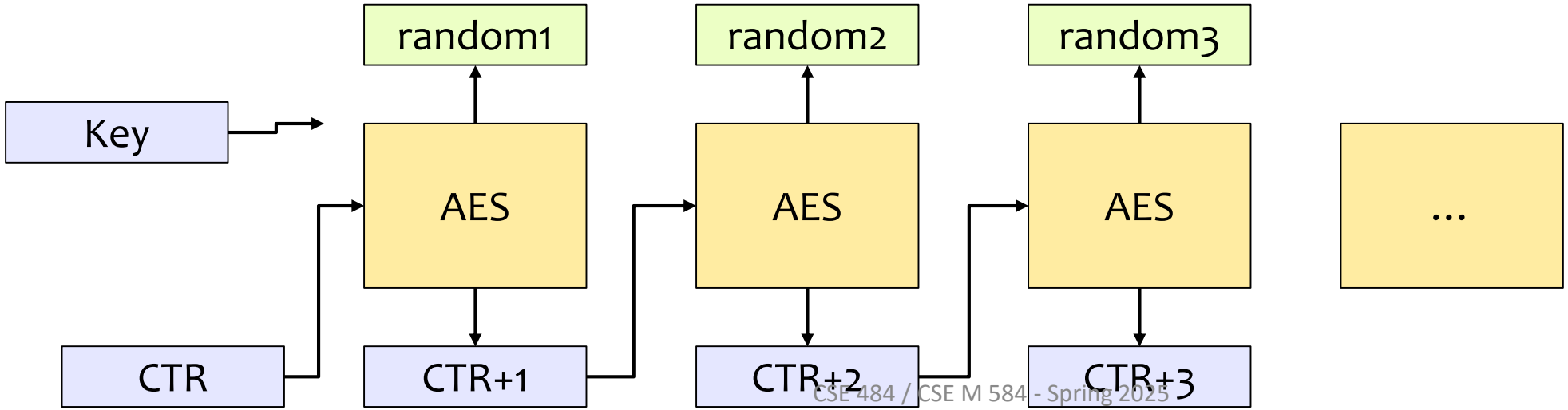
- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)

- We've seen something that works like this for a computationally-bounded attacker!



# AES CTR DRBG

- AES Counter-mode Deterministic Random Bit Generator
- Gather N (say, 1000) random bits.
  - Hash them into two (distinct) 128-bit hashes
  - Key = Hash(bits[0:499])
  - CTR = Hash(bits[500:999])
- Mix in (e.g. CTR = Hash(CTR_current || randombits) regularly.



# Obtaining Pseudorandom Numbers

- Linux:
  - /dev/random – blocking (waits for enough entropy)
  - /dev/urandom – nonblocking, possibly less entropy (??)
  - `getrandom()` – by default, blocking
- Windows:
  - Various things, RNGCryptoServiceProvider maybe
- Challenges with embedded systems, saved VMs

# Obtaining *Random* Numbers

- AMD/Intel's on-chip random number generator
  - RDRAND

- Internally:
  - Entropy pool gathered from multiple sources
    - e.g., mouse/keyboard/network timings

- Hopefully no hardware bugs!
  - https://arstechnica.com/gadgets/2019/10/how-a-months-old-amd-microcode-bug-destroyed-my-weekend/
  - Or only a few bugs?

# Cloudflare LavaRand



https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/