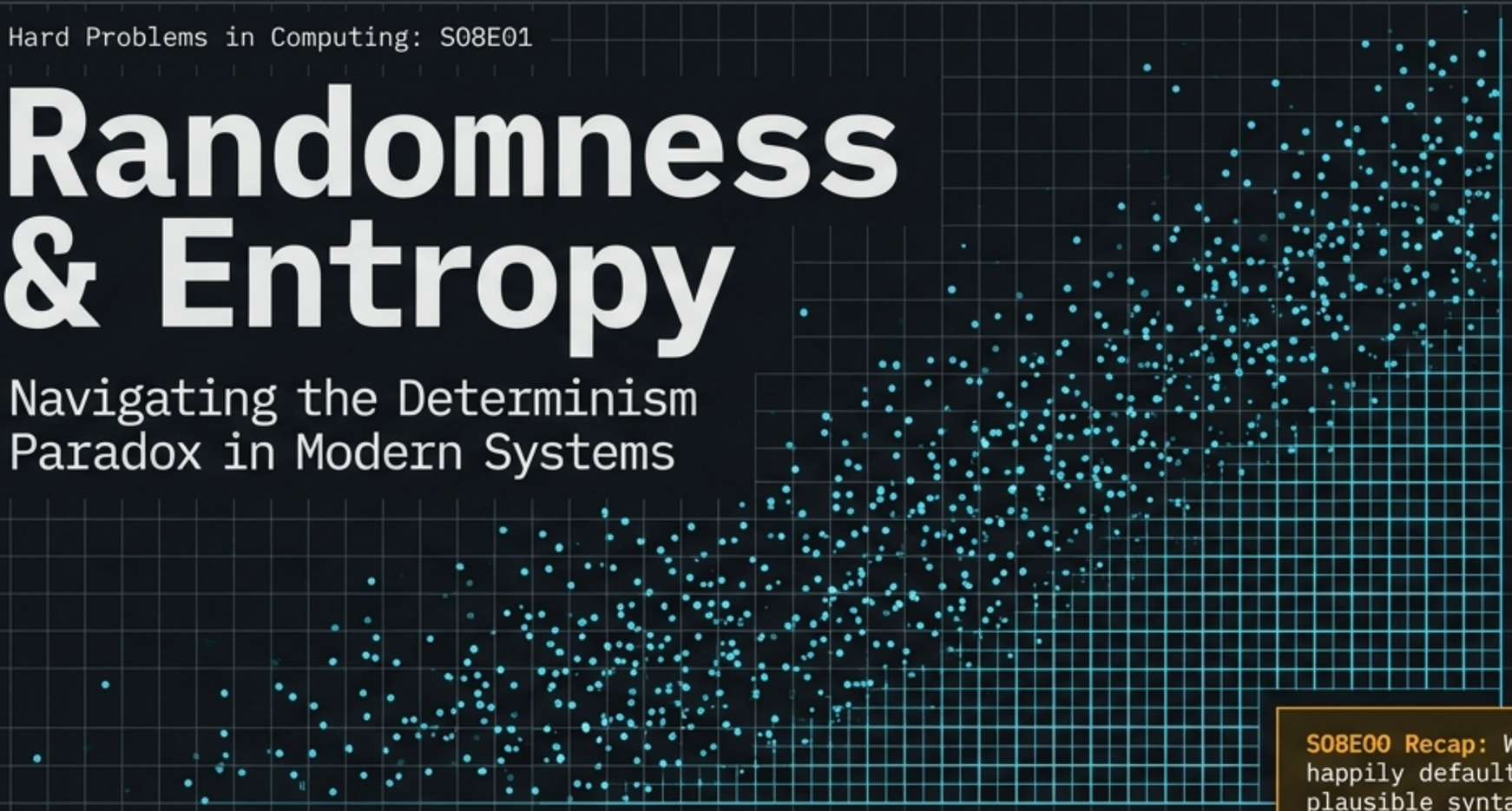
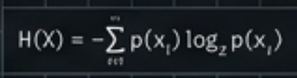
Hard Problems in Computing: S08E01

Randomness & Entropy

Navigating the Determinism Paradox in Modern Systems



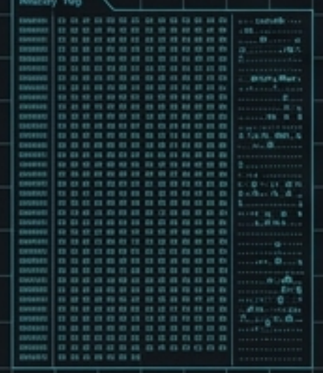


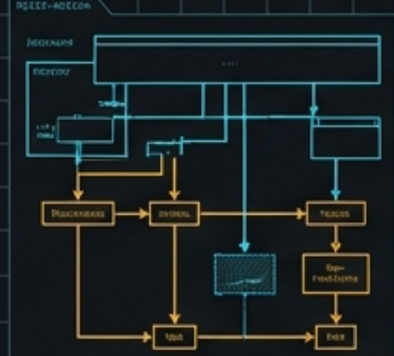


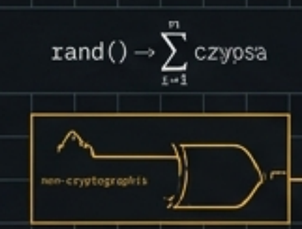


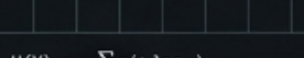


S08E00 Recap: While LLMs happily default to the most plausible syntax on the “happy path,” our Arc 1 journey begins where these statistical autocomplete engines fail hardest—starting with the invisible knots of cryptographic determinism.

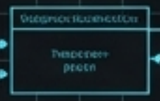










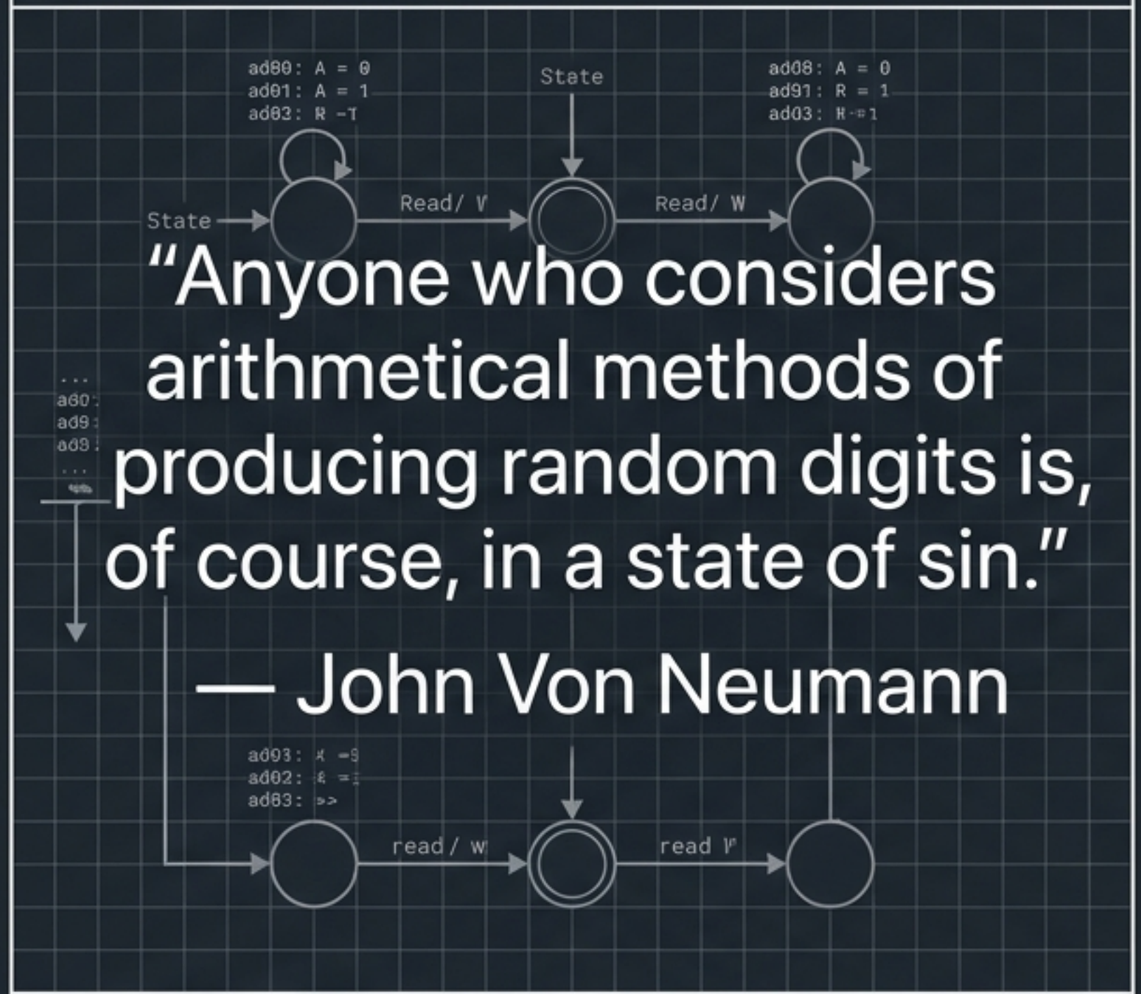






Gemini Notebook

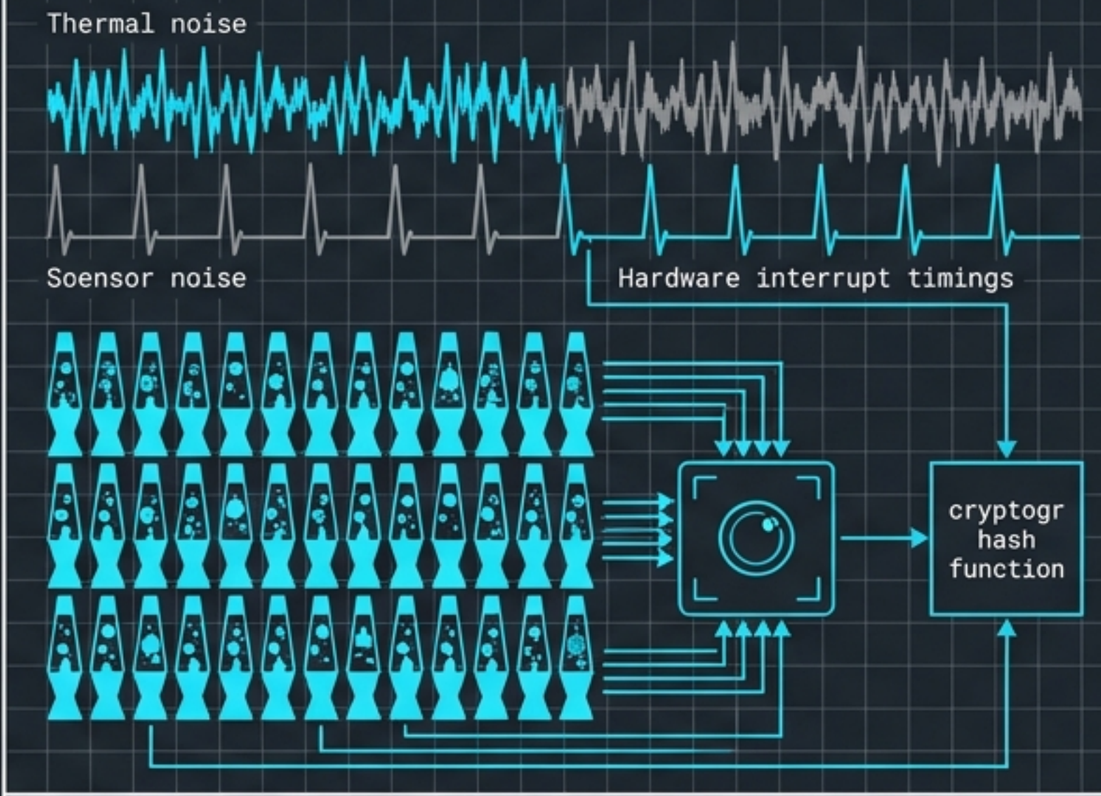
THE LIMIT



THE STATE MACHINE PARADOX

Von Neumann architectures execute deterministic math. They cannot natively flip a coin.

THE ESCAPE



HARVESTING CHAOS

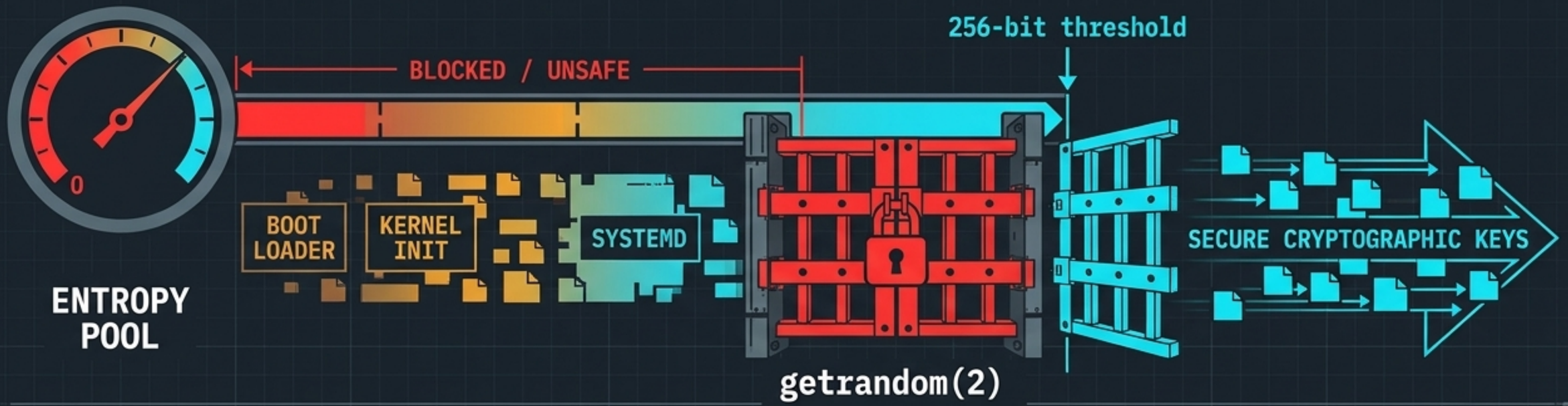
To break determinism, systems must ingest unpredictable physical phenomena.

LAVARAND IN PRODUCTION

Cloudflare digitizes fluid dynamics from 100 lava lamps, feeding unpredictable camera noise into a cryptographic hash to seed production servers.

Gemini Notebook

VM / CONTAINER BOOT SEQUENCE



THE CLONE PROBLEM

VMs and containers boot with identical initial memory states. Without hardware entropy, **cloned environments generate identical SSH/TLS keys**.

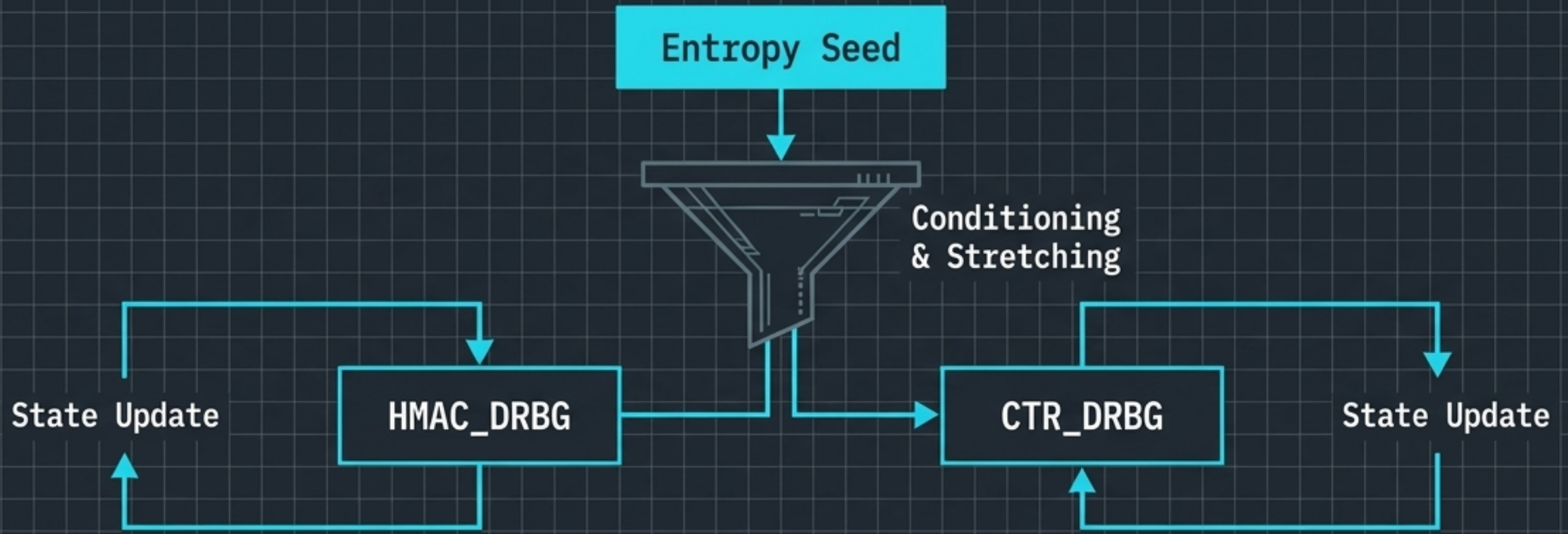
EARLY-BOOT STARVATION

Headless devices **lack interrupt noise** (no mouse, zeroized ECC RAM, predictable clocks). Factorable.net research exposed **mass key duplication**.

THE KERNEL'S DEFENSE

getrandom(2) acts as a safety floodgate. It strictly blocks execution until the kernel pool hits a **safe cryptographic threshold** (~256 bits), preventing identical-key collapse.

Gemini Notebook



NIST SP 800-90A Anatomy

We do not read slow, raw physical noise directly. We use it to seed a Deterministic Random Bit Generator **(DRBG)**.

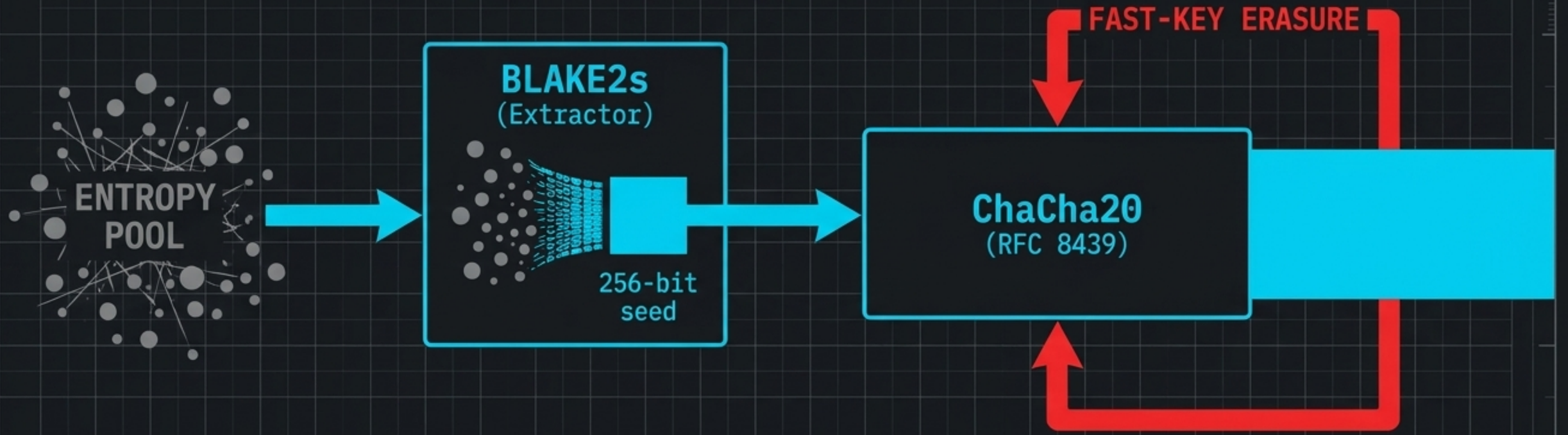
Backtracking Resistance

Even if the current state is compromised, past outputs remain mathematically isolated and **secure** (via one-way hashes).

Prediction Resistance

If the state is compromised, the generator constantly **reseeds** from the physical entropy pool, **cutting off future predictability**.

Gemini Notebook



EXTRACTION (BLAKE2s)

Cryptographic hashing distills messy, biased environmental noise into a mathematically pure, uniform 256-bit seed.

EXPANSION (ChaCha20)

A highly optimized stream cipher takes the seed and generates an endless, cryptographically secure byte stream.

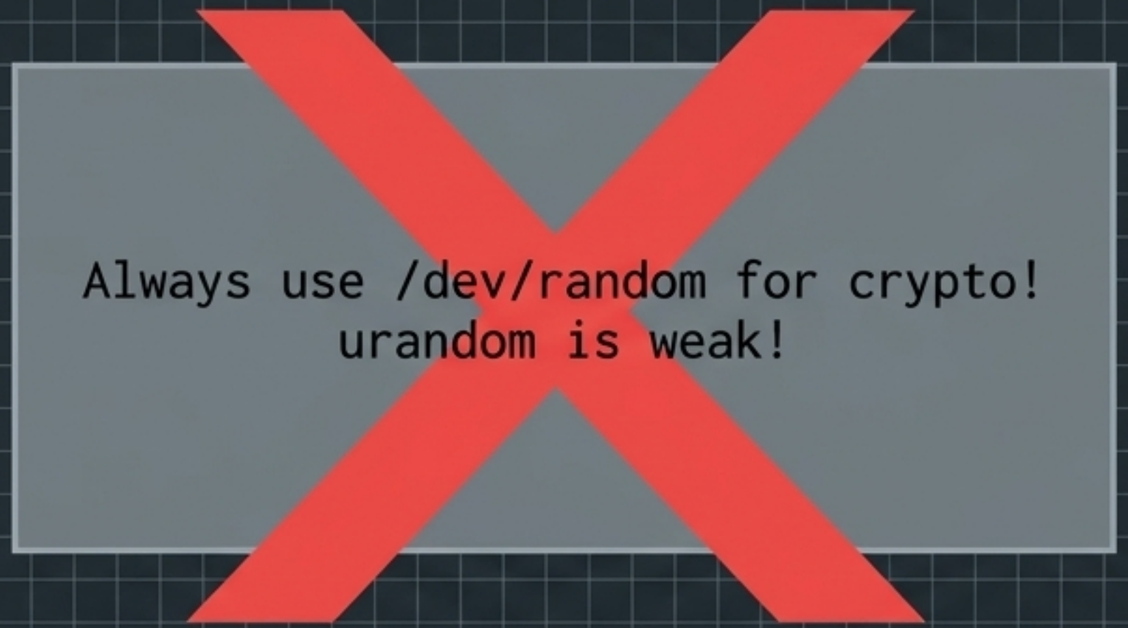
FAST-KEY ERASURE

ChaCha20 uses a portion of its own output to overwrite its internal key. Even if OS memory is dumped, the PRNG state cannot be reverse-engineered.

Gemini Notebook

CRYPTOGRAPHER'S DIAGNOSTIC WORKBENCH

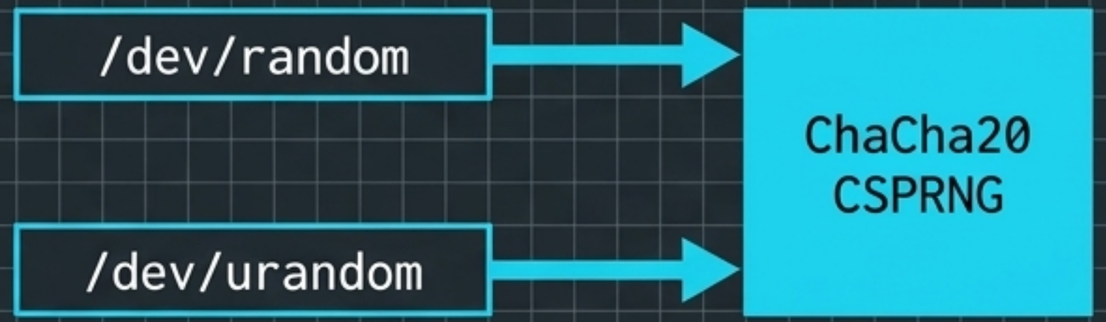
THE MYTH



The Myth: /dev/random provides "true" randomness; /dev/urandom is pseudo-random and weak.

The Danger: Legacy /dev/random blocked when the kernel arbitrarily decided it "ran out of entropy," causing devastating, unpredictable application hangs (a known **DoS vector**).

THE REALITY



The Reality: Both interfaces use the exact same CSPRNG under the hood. 256 bits of initial entropy is enough to **securely** generate near-infinite output.

The Modern Default: Use /dev/urandom (or ideally, getrandom(2) which safely blocks exactly once at boot).

Gemini Notebook

MT19937 State Array (624 32-bit ints)









Untempering Transform



Perfect State Clone

Mersenne Twister (Python random)

Boasts excellent statistical properties and a massive period length (2^19937-1), but it is cryptographically trivial.

The Untempering Attack

By capturing exactly 624 sequential outputs, an attacker can invert the bitwise tempering function, clone the internal state, and perfectly predict all past and future outputs.

Gemini Notebook

CRYPTOGRAPHER'S DIAGNOSTIC WORKBENCH

Algorithm	State Size	Speed	Stat Quality	Predictability	Default Uses
LCG (C rand())	Tiny	Ultra-fast	Poor	Trivially predictable	Legacy/Avoid
MT19937 (Python random)	2KB	Moderate	Good	Perfectly predictable (624 outputs)	General Math
xorshift128+ (V8 Math.random)	Small	Fast	Sub-par	Predictable	Web/UI logic
PCG / xoshiro	128/256-bit	Blazingly fast	Excellent	Predictable (Harder)	Fuzzing / Simulations
OS CSPRNG (Web Crypto, node:crypto, Python secrets)	Large	Slower	Cryptographically Secure	Unpredictable	Secrets / Keys / Tokens

Gemini Notebook

Hall of Fame RNG Outages: Early Naive Hacks

Vibe-Coder Era







Netscape SSL (1996)

Seeded the RNG using Time of Day + PID + Parent PID.

```
srand(time(0) ^ pid ^ ppid);
```

Attackers guessed the PID and time to flawlessly decrypt SSL traffic.

ASF Online Poker (1999)

Shuffled 52! possible decks using a 32-bit LCG.

```
state = (state * 214013 +
25310111) & 0xFFFFFFFF;
```

32 bits only yields 4.2 billion shuffles. Hackers mapped internal states in real-time to know every hand.

Russian Slot Machines (2000s)

Hackers reverse-engineered the PRNGs, mapped the internal state.

and used smartphone apps synced to the slots' timing to predict the exact millisecond of winning pulls.

Gemini Notebook

```
E: y^2 = x^3 + ax + b (mod p)
P = (x_p, y_p)
Q = (x_p, y_p)

Q = d * P
P = (x__s, p)
Q = d * P
s_i = x(s_{i-1} * P)
s_@ = p  * Q
□ = (x- p) + Q - (p * P)

Q = d * P
s_i = x(s_{i-1 * P)
s_i = x(s{i-1 * P)
```

NSA's Dual_EC_DRBG

A NIST-approved PRNG based on elliptic curves. The points P and Q were deliberately chosen by the NSA. Knowing the secret mathematical relationship allowed them to predict future outputs from a single 32-byte sample, breaking SSL/TLS.



USENIX 2012: Mining Ps and Qs

Researchers factored RSA keys for millions of routers. Embedded devices booted with zero entropy, generated identical SSH/TLS states, and resulted in shared prime factors (GCD factorization).

Gemini Notebook

Valgrind Warning: Use of Uninitialized Memory

```
MD_Update(&m,buf,j);
MD_Update(&m,buf,j);  /* purify complains */
```

The Debian OpenSSL Catastrophe (CVE-2008-0166)

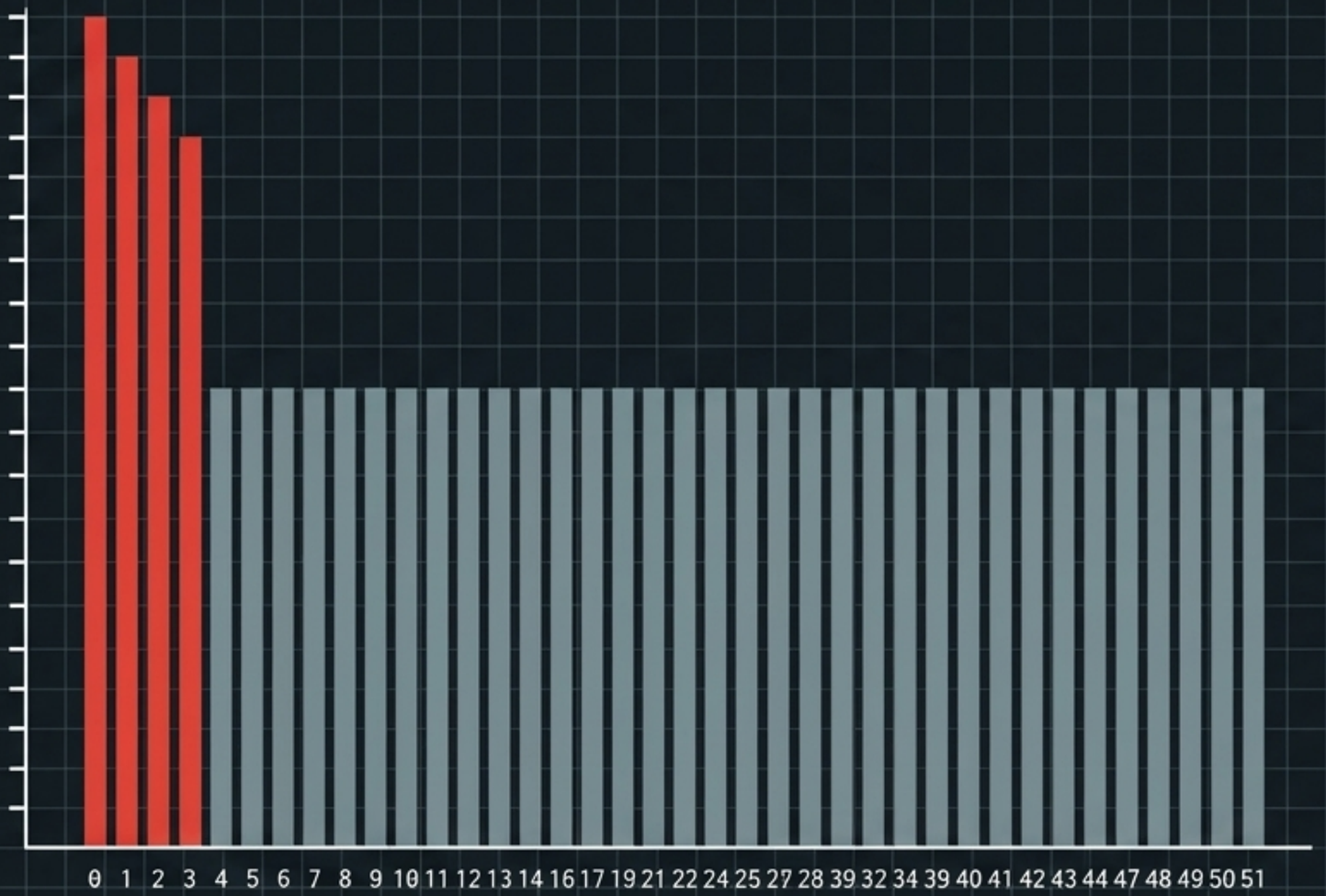
The Context: In 2006, a maintainer ran Valgrind on OpenSSL, which flagged uninitialized memory in the RNG as a bug.

The ‘Fix’: The maintainer deleted the lines mixing the uninitialized buffer into the entropy pool, effectively blinding the PRNG.

The Result: The only remaining entropy was the Linux PID (max 32,768). For for two years, Debian systems generated only 32,768 possible SSH/SSL keys. Total, invisible cryptographic collapse.

Gemini Notebook

2^32 mod 52 != 0



The Flaw

Generating a random integer via `rand() % bound` introduces catastrophic bias.

The Math

If a PRNG outputs 0 to 2^32-1, and the bound is 52 (a deck of cards), 2^32 is not evenly divisible by 52.

The uneven remainder causes the numbers 0-3 to over-index compared to the rest of the set.

The Solution

Rejection sampling. Discard PRNG outputs that fall into the uneven remainder range and roll again.

Securely implemented via `crypto.randomInt()` in Node.js.

Gemini Notebook

THE BIAS: NAÏVE SWAPS VS. UNIFORMITY



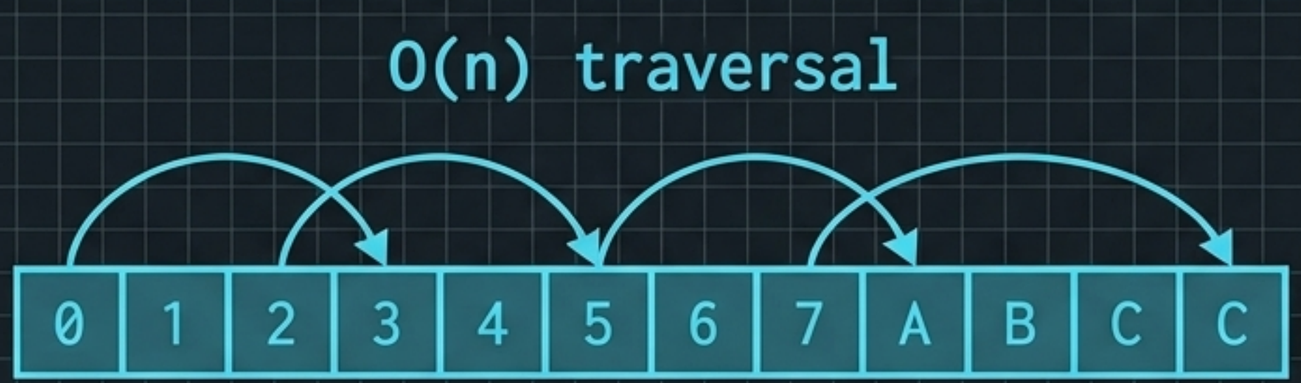

```
array.sort(() => Math.random() - 0.5)
```

Produces **n^n** paths.

For 3 items, **3^3 = 27** paths.

Since there are only **3! = 6** valid permutations, and 27 is not divisible by 6, certain permutations will appear significantly more often.

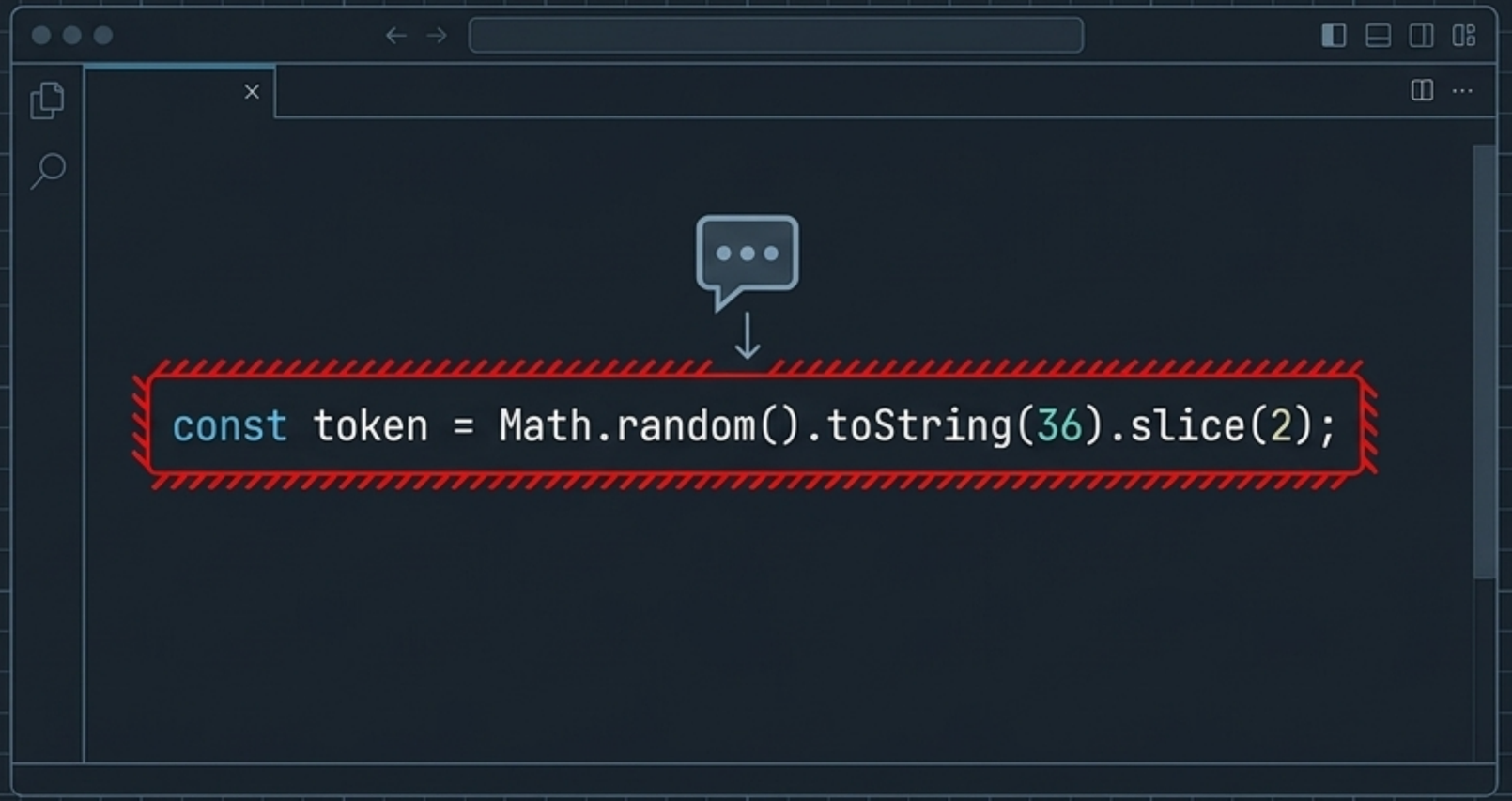
THE SOLUTION: DURSTENFELD FISHER-YATES SHUFFLE



Traverses the array exactly once (`O(n)`).

It swaps each element with a randomly chosen unvisited element, ensuring mathematically uniform distribution without modulo bias.

Gemini Notebook



- **CWE-338:** Use of Cryptographically Weak PRNG
- **The Trap:** LLMs are trained on millions of outdated UI tutorials. They instinctively reach for

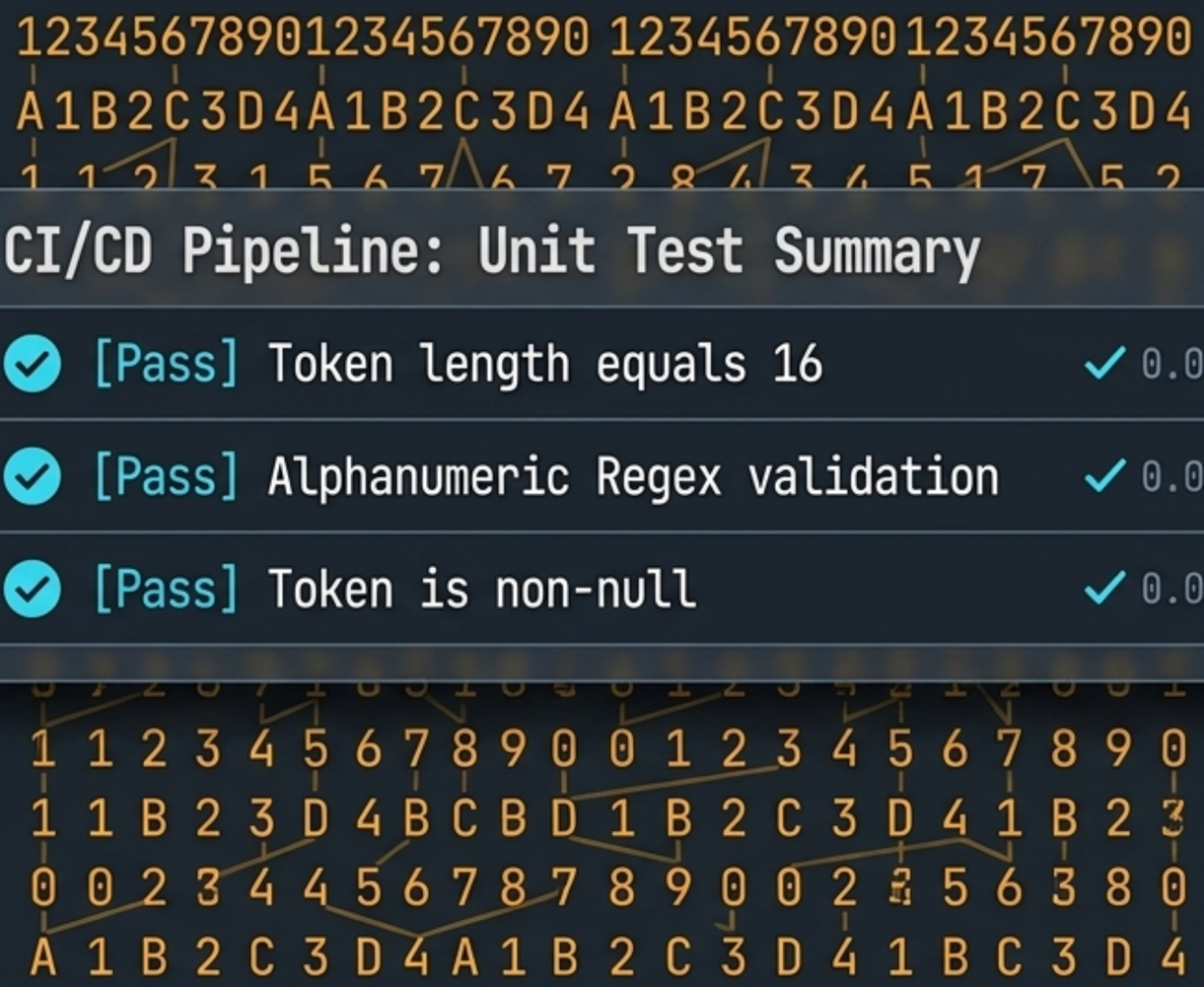
- **The Trap:** LLMs are trained on millions of outdated UI tutorials. They instinctively reach for Math.random() to generate invite tokens, OTPs, and password reset salts.

- **The Impact:** As seen with V8's MWC1616/xorshift, these tokens are statistically weak and trivially predictable, leading directly to session hijacking and account takeovers (e.g., CVE-2021-3692).

Gemini Notebook

Broken PRNG Sequence (e.g., LCG)

Deterministic, Low Entropy



CI/CD Pipeline: Unit Test Summary

[Pass] Token length equals 16	0.02s
[Pass] Alphanumeric Regex validation	0.01s
[Pass] Token is non-null	0.02s

The Invisible Vulnerability

Predictability is invisible in small samples.

A token generated by Math.random() looks exactly as random to human eyes and regex tests as one generated by crypto.randomBytes().

The Testing Paradox

- You cannot write a standard unit test to catch broken entropy.
- The vulnerability survives CI/CD pipelines flawlessly until it hits production and is exploited.

Gemini Notebook

CLAUDE.md / AGENTS.md

```
# Cryptographic Hygiene Rules

1. AST Lint Bans: Explicitly instruct agents to ban Math.random()
(and equivalent weak PRNGs) in security contexts using AST linters.

2. Adversarial/Secrets Workloads:
    MUST use OS CSPRNG (crypto.randomBytes, crypto.randomInt).

3. Deterministic/Simulation Workloads:
    Custom crypto is an anti-pattern. YAGNI dictates using
    explicitly injected, non-global seeded PRNGs (e.g., PCG, xoshiro)
    for fuzzing/simulations to guarantee reproducible test failures.
```

Gemini Notebook

Arc Map



Synthesis

We cannot take the infrastructure of randomness for granted. Chaos is a

physical resource, its depletion causes catastrophic failure, and LLM “happy paths” inherently trend toward insecure determinism.

Next Up in S08E02

We tackle another intractable distributed systems illusion: Idempotency & “Exactly-Once” Delivery. Join us as we dissect network partitions, the Two Generals Problem, and the myth of perfect state synchronization.

Gemini Notebook