# Building a True Random Number Generator (TRNG)

## Technical Approach & Execution Plan

ARUN ALEX

MOHAMMED FATAH

VAMSHI VAVILLA

SIMON SOLIS

OCTOBER 30, 2025

CSE 4392: SPECIAL TOPICS

# What the Project is About (Recap)

- Designing a True Random Number Generator that extracts randomness from fish movements in an aquarium captured via a camera.

# What is a True Random Number Generator?

- Definition:
  A True Random Number Generator (TRNG) derives randomness from physical, unpredictable phenomena in the real world rather than from mathematical algorithms.

- Key Characteristics:

Entropy Source is Physical, not computed (e.g., noise, motion, timing jitter, natural systems).

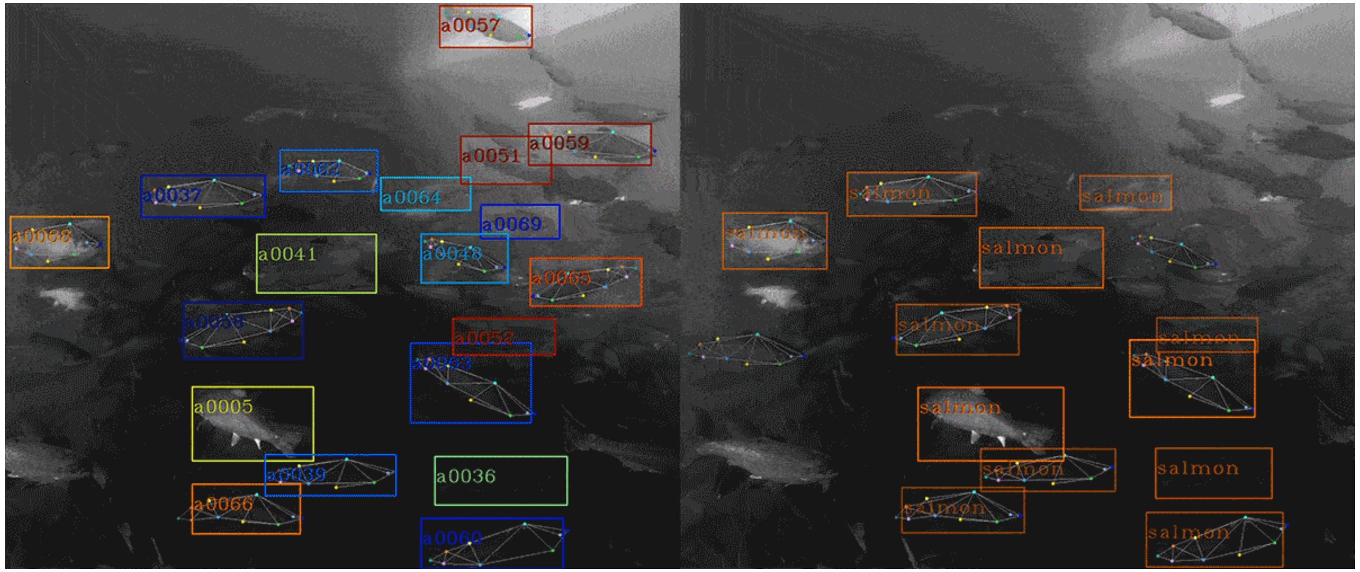
Non-Deterministic — output cannot be reproduced, even if initial state is known.

Unaffected by Seed Predictability — unlike PRNGs (which produce repeatable sequences).

Used in Security-Critical Systems such as encryption, authentication, and key generation.

# What the Project is About

- Entropy Source: Unpredictable paths, speeds, and interactions of fish in water create chaotic patterns (combining biological behavior with fluid dynamics).
- Process: Software streams the live YouTube video feed; image processing tracks the locations of a few fish (e.g., their x-y coordinates in the frame) and compares their relative positions (e.g., distances or angles between them) alongside analyzing pixel changes from their movements to generate random bits.



# TRNG vs PRNG (Pseudorandom number generator)

| Aspect | TRNG (True) | PRNG (Pseudo) |
|---|---|---|
| Source | Physical world (nature/hardware) | Algorithm + initial seed |
| Predictability | Unpredictable | Predictable if seed is known |
| Reproducibility | Not repeatable | Repeatable sequence |
| Security Use | High-security cryptographic systems | Simulation, games, non-secure RNG |

# Problem Overview / Motivation
# Why randomness matters in cryptography/security

- Cryptographic systems rely on randomness to generate secure keys, nonces, session tokens, and authentication challenges.
- If the randomness used is predictable, the entire security system collapses, regardless of the encryption algorithm strength.
- Entropy directly determines key strength. A 256-bit AES key with only 40 bits of true entropy is effectively a 40-bit key, regardless of algorithm strength.
- Attackers do not break encryption first — they break randomness.

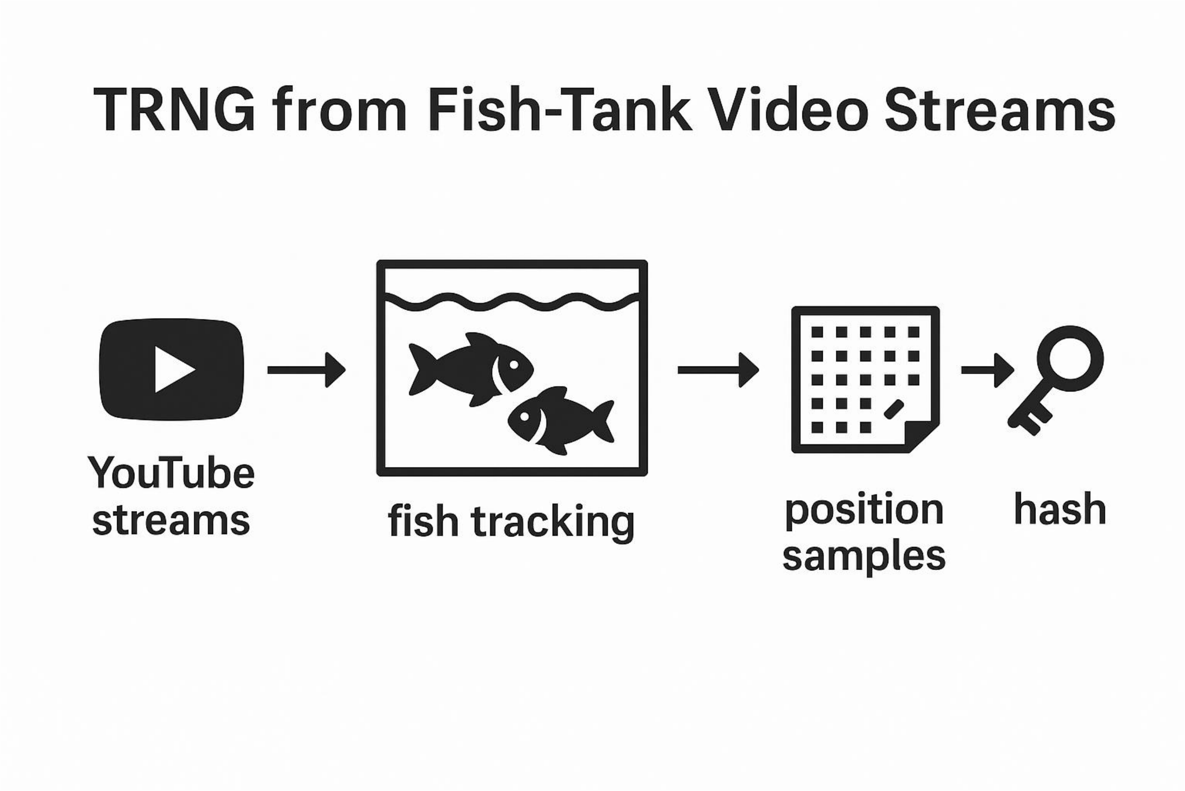
# Weakness of PRNGs (Pseudo-Random Number Generators) in secure systems

- PRNGs use deterministic algorithms initialized with a seed.
- If the seed is guessed, leaked, or low-entropy, an attacker can reconstruct all past and future outputs.
- Many systems unintentionally reuse or weakly derive seeds, leading to catastrophic key reuse.

# Real-world attacks exploiting weak random sources

| Incident | Failure Mode | Impact |
|---|---|---|
| Debian OpenSSL Bug (2008) | Removed entropy-gathering code → RNG had only 15 bits of entropy | Millions of SSH keys globally were cloneable |
| Sony PlayStation 3 ECDSA Hack | RNG reused the same k value in signatures | Private key for Sony firmware signing was exposed |
| Android Bitcoin Wallet Hacks (2013) | Java SecureRandom failed on early Android → repeated nonce reuse | Attackers recovered private blockchain keys and stole BTC |
| Netscape SSL (1996) | RNG seeded with time + PID only | SSL sessions decryptable by attackers |

# TRNG from Fish-Tank Video Streams



- Overall Goal
- Extract unpredictable entropy from positions of fish in live YouTube streams and convert it into cryptographic keys.
- Input/Output
- Input = 2–3 YouTube streams (rotating)
- Output: hashed keys (e.g., 256-bit)
- Flow of data
- Video capture → Fish tracking → Position data → Hash → Key.

# Technical Overview

- System Pipeline
- Fetch frames from 2–3 YouTube streams (rotating).
- Preprocess: resize / color convert / stabilize.
- Detect fish and compute centroids (x, y).
- Select a *random subset* of fish per sampling epoch.
- Quantize/time-stamp values → produce raw bit blocks.
- Condition via a cryptographic hash / extractor → produce key material.
- Randomness tests (NIST ) and entropy estimation.

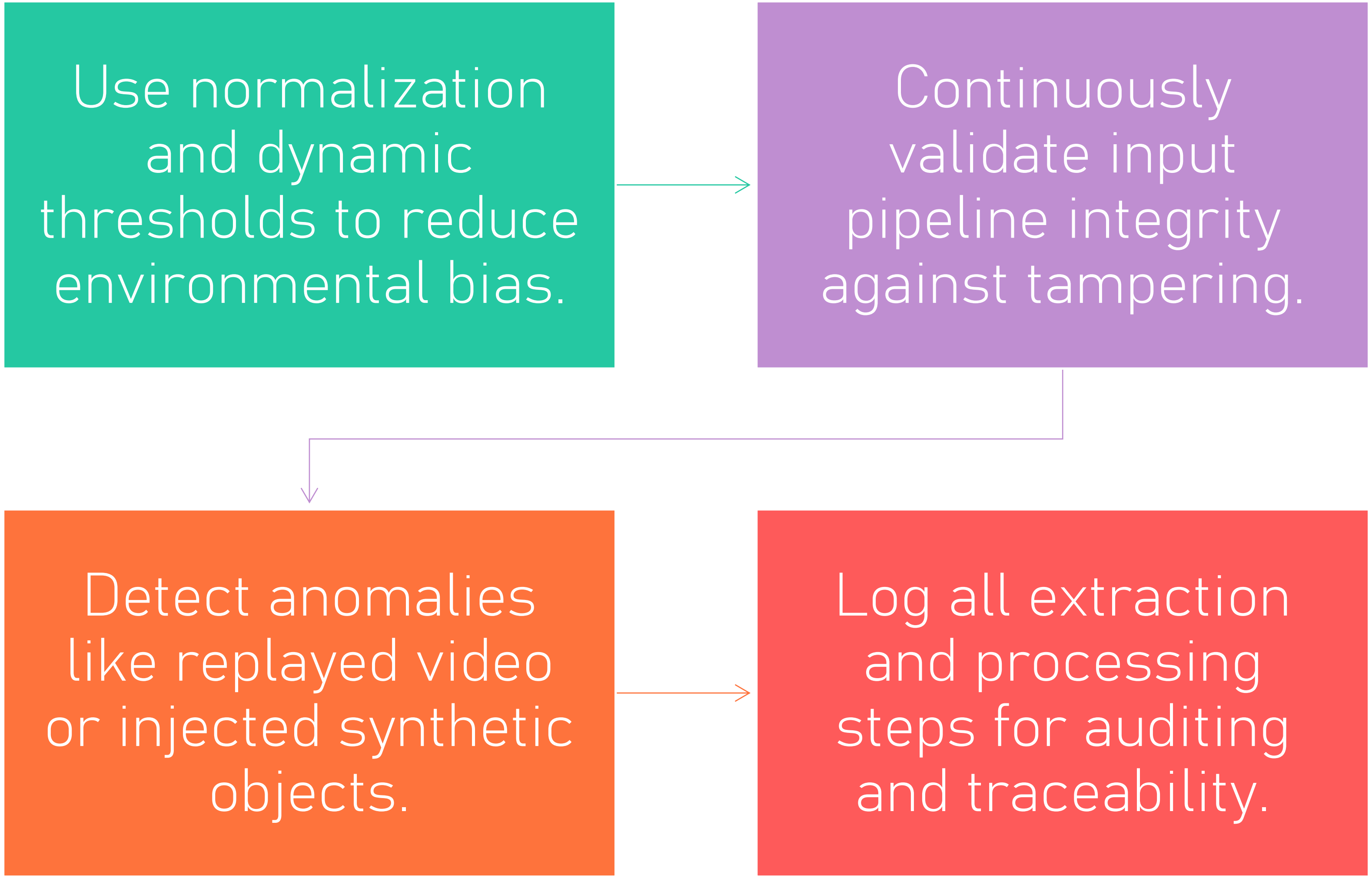
# Technical Overview

- Quantization & Selection
- positions → bits
- Quantization: map (x,y) pixel coords → fixed integer range (e.g., 0..1023 → 10 bits each).
- Concatenation: `bits = [timestamp || id || xbits || ybits || area_bits]`.
- Random subset selection: use reservoir sampling or shuffle index each epoch so the set of fish used changes.
- Windowing: accumulate N samples per key generation (to reach desired entropy).
- Balance between too coarse (loss of entropy) and too fine (overfitting to noise) quantization.

# Technical Overview

- Hashing & Key Derivation
- remove bias & correlations, prevent attacker exploitation.
- Use cryptographic hash (SHA-256) to “whiten” concatenated samples.
- Options for multiple keys: feed hash output into HKDF to derive multiple keys.
- Entropy estimate: Before trusting bits, estimate min-entropy per sample
- If entropy is low, increase number of samples per key or use more streams.
- Concentrating existing entropy safely.

# Technical Overview



Use normalization and dynamic thresholds to reduce environmental bias.

Continuously validate input pipeline integrity against tampering.

Detect anomalies like replayed video or injected synthetic objects.

Log all extraction and processing steps for auditing and traceability.

# Technical Overview

- Periodically update fish tracking algorithms to improve detection accuracy.
- Integrate redundancy by sampling overlapping video segments for higher entropy.
- Automate NIST randomness and entropy tests for regular compliance.
- Implement real-time alerts when entropy metrics fall below secure thresholds.

# Threat Model Overview

Adversaries

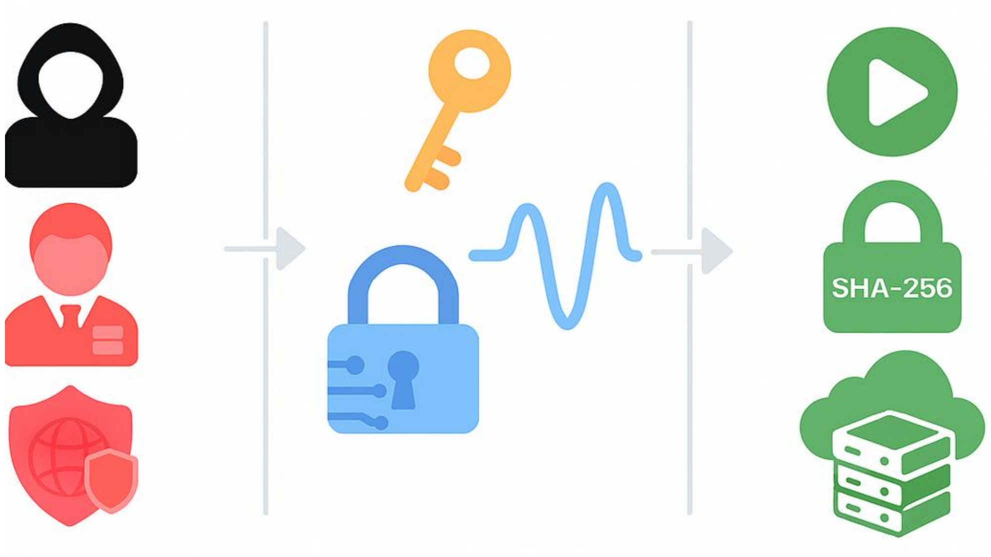
- Remote attacker (internet-based, no physical access).
- Insider attacker (access to software pipeline or system logs).
- Advanced adversary (state-level, with resources to manipulate entropy sources).

Assets to Protect

- Randomness quality (entropy strength).
- Generated cryptographic keys.
- Integrity of entropy pipeline (video input → key output).

Assumptions

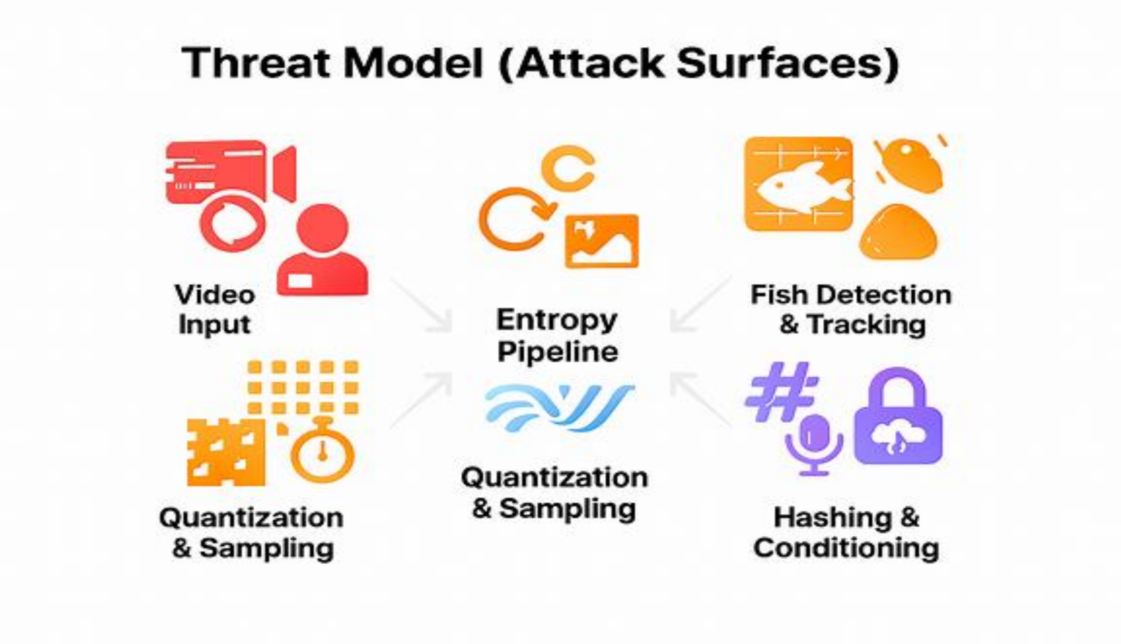
- Public video streams are not under attacker’s direct control.
- Cryptographic primitives (SHA-256, HKDF) are secure.
- System environment is not fully compromised.



# Threat Model: Attack Surfaces

Potential Points of Exploitation:

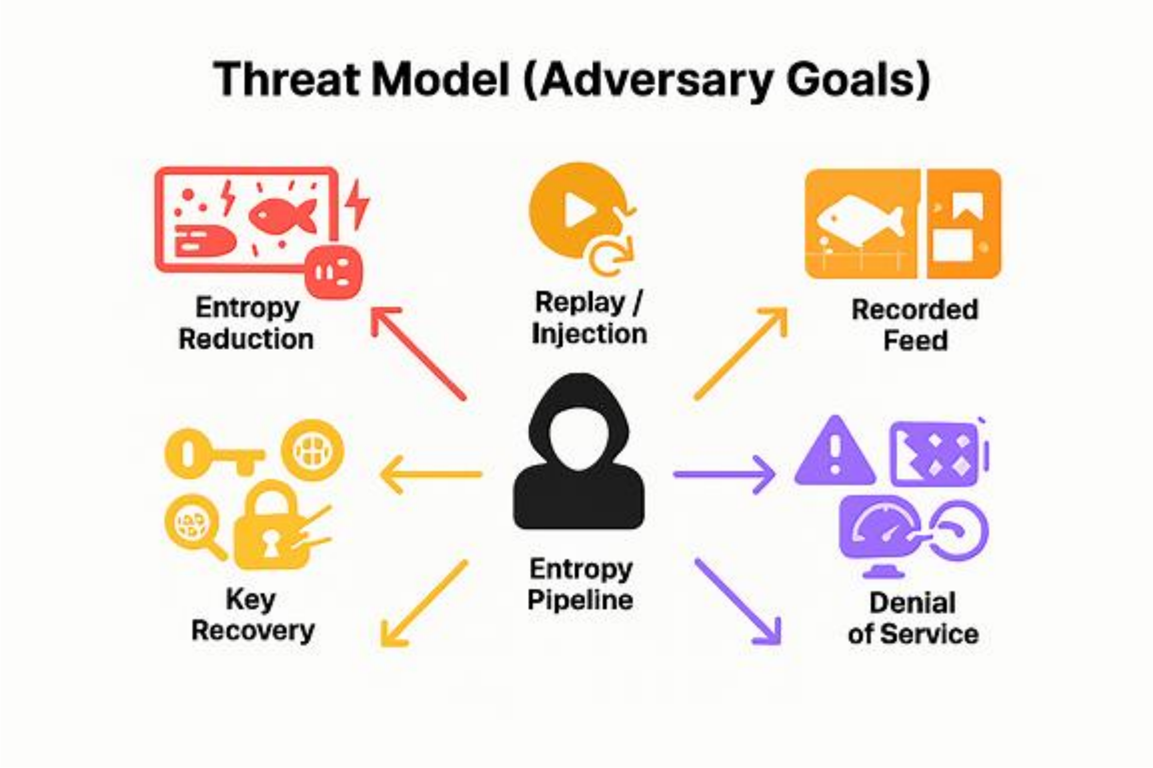
- Video Input
  - Stream tampering (injected or replayed video).
  - Frame drops or synchronization attacks.
- Fish Detection & Tracking
  - Adversary introduces artificial patterns (e.g., overlayed graphics, lighting manipulation).
  - Misclassification of objects as "fish" → biased entropy.
- Quantization & Sampling
  - Predictable mapping if quantization too coarse.
  - Correlation across frames if sampling rate too high.
- Hashing & Conditioning
  - Weak or omitted conditioning could leak bias.
  - Side-channel leakage of pre-hash values.



# Threat Model: Adversary Goals

What an Attacker Wants

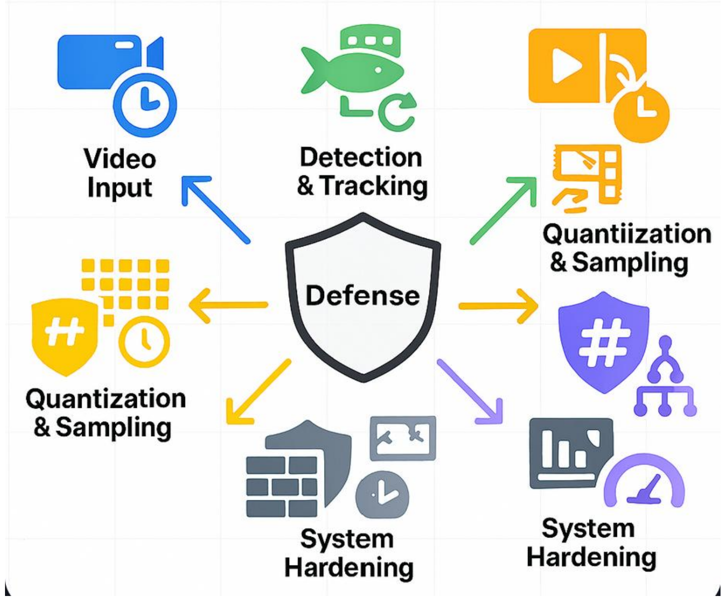
- Entropy Reduction
  - Bias fish movement patterns (e.g., flashing lights, feeding schedules).
  - Force predictable trajectories → reduce randomness.
- Replay / Injection
  - Replace live feed with pre-recorded video.
  - Replay known fish patterns to predict outputs.
- Key Recovery
  - Exploit low-entropy outputs to brute-force derived keys.
  - Correlate multiple outputs to infer internal state.
- Denial of Service
  - Overwhelm system with corrupted frames.
  - Cause entropy estimator to fail → no keys generated.



# Threat Model: Mitigations & Defenses

How We Defend Against Threats

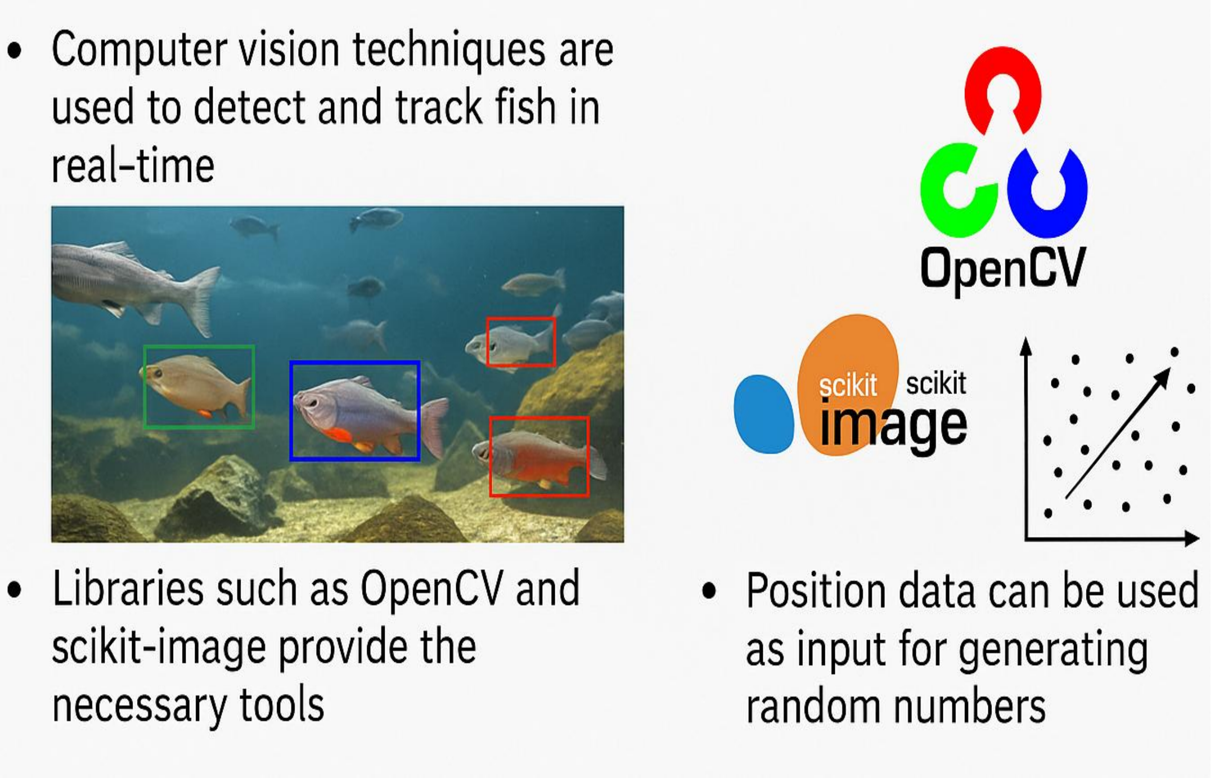
- Video Input
  - Use multiple independent streams (cross-entropy).
  - Timestamp verification to detect replay.
- Detection & Tracking
  - Confidence thresholds: discard low-certainty detections.
  - Rotate fish subsets randomly each epoch.
- Quantization & Sampling
  - Fine-tuned quantization to balance entropy vs. noise.
  - Randomized sampling intervals to prevent correlation.
- Hashing & Conditioning
  - SHA-256 whitening removes bias.
  - HKDF for multiple key derivations.
- System Hardening
  - Drop samples on frame loss or low entropy.
  - Log metrics for anomaly detection.



# Threat Model: Residual Risks & Assumptions

- Remaining Risks
  - If all input streams are compromised, entropy collapses.
  - Biological systems may have hidden correlations (e.g., schooling behavior).
  - Extreme adversaries could manipulate aquarium environment.
- Assumptions
  - At least one entropy source remains independent.
  - Cryptographic primitives remain secure.
  - System is deployed in a trusted execution environment.

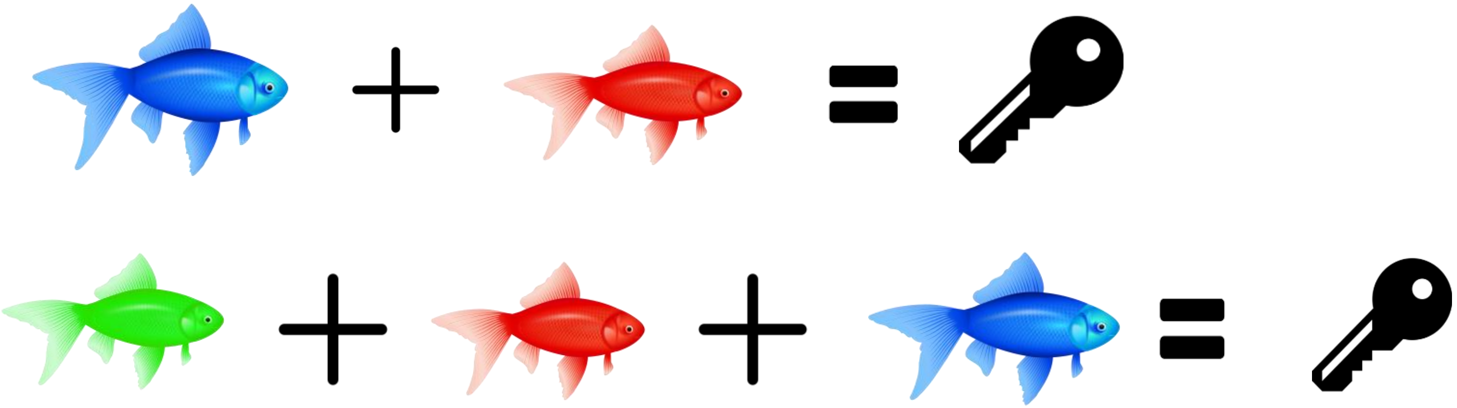
# Execution Plan



- Computer vision techniques are used to detect and track fish in real-time
- Libraries such as OpenCV and scikit-image provide the necessary tools
- Position data can be used as input for generating random numbers

- Video Input & Capture
- Libraries: ffmpeg, opencv (cv2), yt-dlp or pytube to obtain stream URLs.
- Use cv2.VideoCapture(stream_url) or ffmpeg subprocess to read frames.
- Target FPS: 5–30 fps (tradeoff: more samples vs correlated frames).
- Sync across streams: use UTC timestamps / monotonic clock.
- Video Input & Capture
- Deep learning object detector (YOLOv8 / MobileNet SSD) trained for fish (higher accuracy).
- Tracking: SORT or DeepSORT to maintain IDs across frames.
- Tracking output: for each fish → `(id, timestamp, x, y, area, speed_vector)`.

# Execution Plan



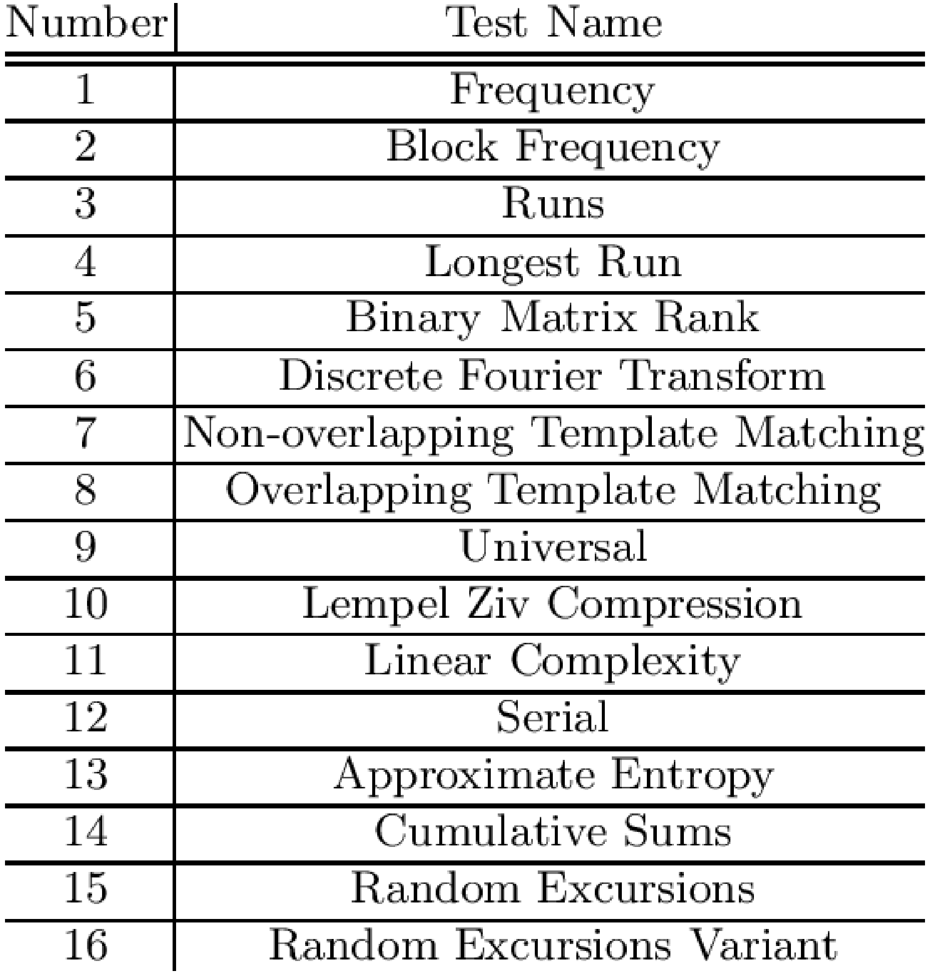
- **Multi-Stream Strategy & Stronger Entropy**
- Use 2–3 independent streams:
  - If streams are independent, combine samples (e.g., XOR or concatenation) — increases entropy.
  - Rotate which streams and which fish are used to prevent long-term correlation.
- Hopefully, **combined streams raises entropy** but need to **look out for sources that share common noise**

# Execution Plan

- Data Collection & Preprocessing Pipeline
- Resize frames to uniform dimensions (e.g., 640×480).
- Filter out stationary fish using minimum speed thresholds.
- Handle network latency and stream desynchronization, fallback to single stream if needed.
- Enable detailed error reporting for all discarded or rejected samples to facilitate troubleshooting and audit trails.

# Execution Plan

- Testing, Metrics & Hardening
- Tests:
  - statistical tests such as NIST SP800-22, Dieharder, ENT (run on final keys).
  - Entropy estimation (min-entropy per bit).
- Hardening:
  - Discard samples when network frames drop or when detection confidence is low.
  - Log metrics (detection rates, sample counts, hash inputs).
- If entropy below threshold, do not issue a key.



| Number | Test Name |
|---|---|
| 1 | Frequency |
| 2 | Block Frequency |
| 3 | Runs |
| 4 | Longest Run |
| 5 | Binary Matrix Rank |
| 6 | Discrete Fourier Transform |
| 7 | Non-overlapping Template Matching |
| 8 | Overlapping Template Matching |
| 9 | Universal |
| 10 | Lempel Ziv Compression |
| 11 | Linear Complexity |
| 12 | Serial |
| 13 | Approximate Entropy |
| 14 | Cumulative Sums |
| 15 | Random Excursions |
| 16 | Random Excursions Variant |

# Execution Plan

- Deliverables
  - Demo video of our program
  - Showing of the single stream vs multi-stream entropy evaluation
  - Run the randomness test and assess the functionality of our implementation

# Questions

# Reference

- https://spectrum.ieee.org/aquaculture
- https://explore.org/livecams/aquarium-of-the-pacific/pacific-aquarium-tropical-reef-camera
- https://doge.tg/blog/2018/The-2013-Android-Bitcoin-Wallet-Vulnerability-A-Lesson-in-Randomness/
- [NIST 2010] Rukhin, A., et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (SP 800-22 Rev.1a).* NIST, 2010. https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
- [Lenstra 2012] Lenstra, A. K., Hughes, J. P., Augier, M., Bos, J. W., Kleinjung, T., & Wachter, C. "Ron was wrong, Whit is right." *IACR ePrint Archive*, 2012. https://eprint.iacr.org/2012/064.pdf
- [Bonneau 2013] Bonneau, J., & Mironov, I. "Blockcipher-based PRNGs and Security Failures." *IEEE Symposium on Security and Privacy (S&P)*, 2013. https://ieeexplore.ieee.org/document/6547124
- [RFC 4086] Eastlake, D. "Randomness Requirements for Security." *RFC 4086*, IETF, 2005. https://www.rfc-editor.org/rfc/rfc4086
- [Barak 2019] Barak, B., et al. "True Randomness from Biological Systems." *Journal of Cryptographic Engineering*, 2019. https://link.springer.com/article/10.1007/s13389-019-00209-9