Raspberry Pi (Pi) **nhận bản mã từ ESP qua LoRa**, sau đó:

1. **Giải mã** bằng ASCON-128 với KEY/AD để lấy **plaintext thật**.
2. **Mã hoá lại** plaintext đó bằng **NEW\_KEY/NEW\_AD** để đưa lên web.
3. **Đo thông lượng (throughput)** theo đúng 2 giai đoạn thực tế:
   * **Stage 1 – Decryption (KEY/AD)**: đo giải mã.
   * **Stage 2 – Re-Encryption (NEW\_KEY/NEW\_AD)**: đo mã hoá lại.

Việc đo chỉ chạy **một lần duy nhất** ở gói hợp lệ đầu tiên; không ảnh hưởng luồng xử lý bình thường.

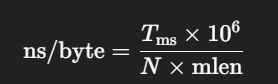
Đo tốc độ lõi mã hoá AEAD (không gồm I/O, không gồm sinh nonce/ngẫu nhiên, không gồm quản lý khung/gói).

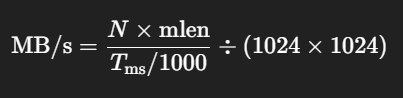
Thông số in ra:

* Time/byte (ns): thời gian trung bình mã hoá 1 byte
* Throughput (MB/s): băng thông (MiB/s) = tổng dữ liệu / thời gian.

Công thức tính thông lượng:

Với kích thước bản tin sau padding là mlen (byte), số vòng đo N, tổng thời gian đo cho một pha là T\_ms (mili-giây):

Time/byte (ns/byte): 

Throughput (MB/s)

T\_ms được đo bằng hàm get\_time\_ms() (CLOCK\_MONOTONIC) bao quanh vòng lặp mã hoá/giải mã.

**Kỹ thuật đo áp dụng:**

Clock ổn định: dùng clock\_gettime(CLOCK\_MONOTONIC) → không bị ảnh hưởng thay đổi hệ thống thời gian.

Warmup: chạy WARMUP\_ITERATIONS để làm ấm cache/branch predictor, loại bỏ “lần đầu chậm”.

Cố định kích thước: dùng dữ liệu thật vừa giải mã, padding tới size cố định (mặc định {1024}).

Loại nhiễu RNG:

* Stage 1 (Decrypt): tạo ct||tag một lần trước khi đo. Vòng đo chỉ gọi crypto\_aead\_decrypt().
* Stage 2 (Re-Encrypt): tạo một base\_nonce ngẫu nhiên, rồi tăng nonce (nonce\_increment) mỗi vòng → không gọi RNG trong vòng đo nhưng vẫn đảm bảo nonce duy nhất.

Tách luồng xuất:

* STDERR: in báo cáo benchmark (khối header + bảng Metric | Value).
* STDOUT: in 3 dòng nonce/ciphertext/tag cho pipeline/web.

→ Không cản trở lẫn nhau; pipeline vẫn chạy “mượt”.

Chạy đúng 1 lần: mỗi benchmark có static int done để chỉ chạy ở gói hợp lệ đầu tiên.

# Giải thích “vì sao nhanh/ổn định”

* **Warm-up + MONOTONIC clock** → bỏ “lần đầu chậm”, thời gian đo không bị hệ thống chỉnh giờ.
* **Không RNG/printf/malloc trong loop** → vòng đo chỉ gồm **khối mật mã**, nên throughput gần sát năng lực thực của thư viện ASCON.
* **Nonce tăng dần** (Stage 2) giữ đúng nguyên tắc AEAD (nonce duy nhất) mà **không** trả giá bằng RNG.
* **Padding cố định & buffer liên tục** → ít branch/ít cache miss, CPU tận dụng cache line tối đa → số đo đều.

# Luồng chạy của chương trình

1. **Khởi tạo & mở UART**
   * Mở /dev/serial0 @115200, 8N1, non-blocking; select() chờ dữ liệu.
2. **Nhận 3 dòng hex**
   * Lần lượt nhận: **nonce** → **ciphertext** → **tag**.
   * Chuyển **hex → bytes**, ghép thành ct || tag.
3. **Giải mã (KEY/AD)**
   * crypto\_aead\_decrypt(m, &mlen, NULL, ct||tag, AD, nonce, KEY)
   * Nếu OK: in các trường cảm biến (stdout).
4. **Benchmark – Stage 1: Decrypt-only** (chạy 1 lần ở gói đầu)
   * Tạo message\_pad từ m bằng padding tới kích thước cố định.
   * **Pre-encrypt 1 lần** để có ct||tag hợp lệ (ngoài vùng đo).
   * **Warm-up decrypt** → **Đo vòng decrypt** → Tính ns/byte, MB/s (in stderr).
5. **Benchmark – Stage 2: Re-Encrypt-only** (chạy 1 lần ở gói đầu)
   * Dùng message\_pad.
   * Tạo base\_nonce (RNG 1 lần), **tăng nonce** mỗi vòng.
   * **Warm-up encrypt** → **Đo vòng encrypt** → Tính ns/byte, MB/s (in stderr).
6. **Mã hoá lại để gửi web (luồng thật)**
   * Tạo new\_nonce bằng getrandom() (**mỗi gói một nonce**).
   * crypto\_aead\_encrypt(…, NEW\_AD, new\_nonce, NEW\_KEY).
   * In 3 dòng (stdout): new\_nonce / ciphertext / tag.
7. **Lặp lại** với gói kế tiếp.

Kết quả:

