Health tests in 90B

John Kelsey, NIST and KU Leuven

Components of an Entropy Source

Noise Source

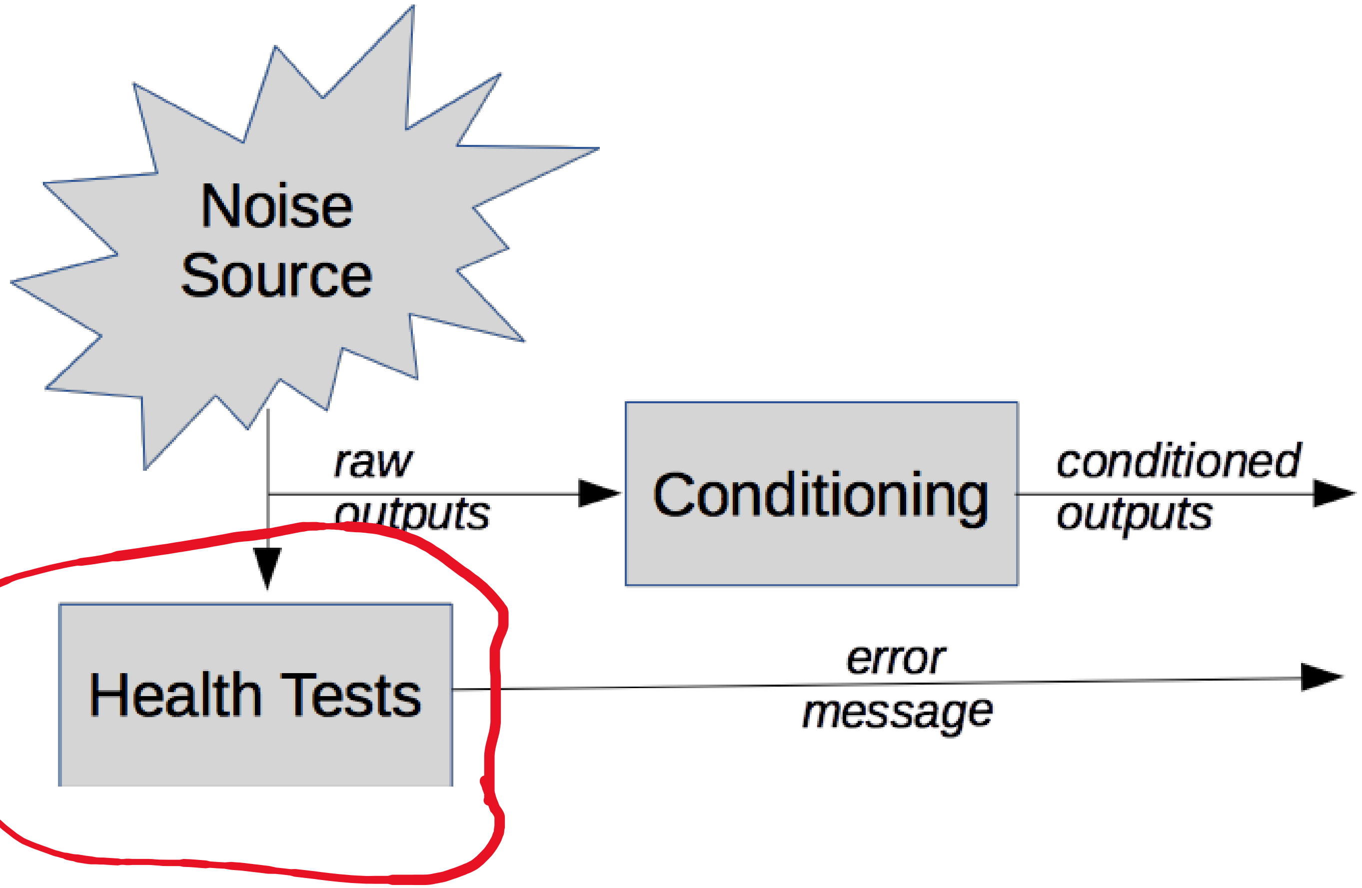
Where the entropy comes from

Health tests

Verify the noise source is still working correctly

Conditioning

Optional processing of noise source outputs before output.



Reminder: An entropy source provides bitstrings with known entropy/sample

Health Tests: Continuous / Startup / On Demand

Continuous Tests

- Going on all the time behind the scenes
- 90B requirements mostly here

Startup Tests

- Run at startup
- May just be continuous tests run over many bits

On Demand

- Run when requested
- May just be rerun of the startup tests

1 Why do entropy sources need health tests?

Noise sources are fragile

- Total failure –oscillators lock together, component fails, etc.

- Very sensitive to internal parameters/layout
  - Parameters outside acceptable range = low entropy
  - Examples: TERO, coherent sampling TRNG

- Many ways for parameters to vary
  - Process variation
  - Environmental variation

Validation process issues

- Lab tests outputs from <u>one</u> sample device
  - At best, maybe a handful

- Successful products may have millions of instances

- Easy for some fraction to be outside acceptable parameter space
  - Process variation, changes to manufacturing, component aging

Critical question: how would we know if ***this instance*** *bad?*

Failures can be subtle

- Known answer tests useless

- Some failures obvious
  - E.g., output stuck on zero

- Many failures more subtle
  - Low entropy output but not obviously bad
  - Test to detect must be specific to entropy source and failure mode

- May need to look at internal values

Noise source failure invisible

- No interoperability problems
  - Everything works fine, just insecure
- Failures masked by conditioning
- Humans never see entropy source outputs
  - Used to seed a DRBG
  - That will mask anything

Entropy sources need health tests!

- Entropy sources can fail undetectably

- Failures can lead to big security problems
  - Attacker guessing private key

- Not just a box checking exercise

failure → signal → detection → reaction

2 Framework

A maybe useful framework

- failure → signal → detection → reaction

- **Failure:** Something causes entropy estimate to be incorrect
- **Signal:** Observable behavior changes
- **Detection:** Health test detects signal
- **Reaction:** Module does something to prevent loss of security

Note: there are many other useful ways to think about health tests

Failure: We need to know what to test for

Failure = entropy estimate no longer valid

- Total failure = catastrophic loss of entropy
  - Example: Oscillator locks to clock
- More subtle failure
  - Parameters out of range for entropy estimate
  - Assumption of entropy estimate falsified
- Environmental conditions
- Attacks

Designer needs to know how entropy claim can be wrong

Signal: Observable change in behavior

failure → signal → detection → reaction

May be a change in statistics of:

- Raw output bits
- Internal values
- Conditioned outputs*

Need to show that failure will lead to signal!

Where do we see the signal?

failure → signal → detection → reaction

- Raw bits
- Conditioned output bits
- Internal values

- 90B says test raw bits
- That's usually right
- Sometimes makes more sense to test internal values
- Occasionally even OK to test conditioned outputs
    - Extra work
    - Need to show you can detect failures through conditioning

Detection: Health test

failure ➡ signal ➡ detection ➡ reaction

Choose health test to detect signal of failure

- Need to consider false positive vs false negative rates

- False alarm: (false positive)
  - Entropy source operating correctly
  - Alarm raised

- Silent failure: (false negative)
  - Entropy source producing less entropy than claimed
  - No alarm raised

False positive/ false negative

Big problem for continuous tests:

- Running all the time
- Potentially processing millions of bits
- 10^{-5} false positive rate → lots of false alarms

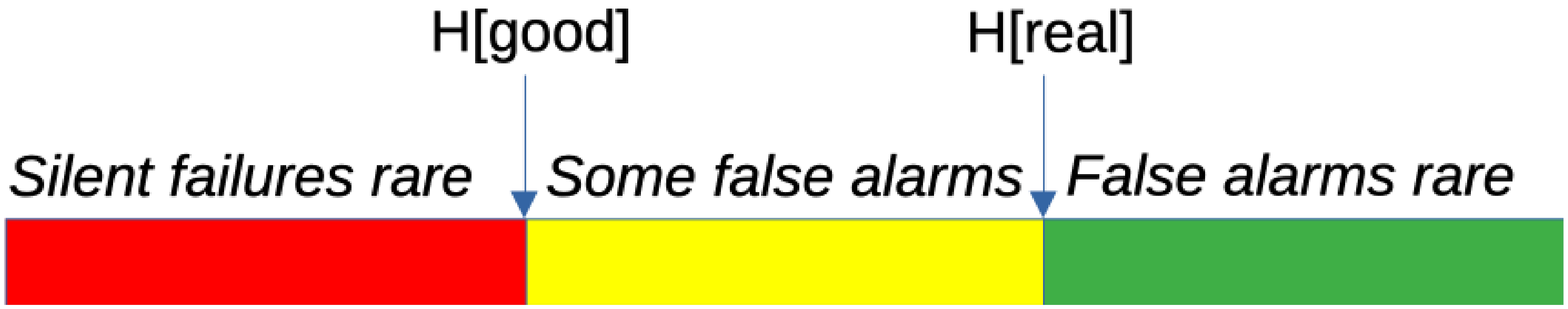
Cutoff values with extremely low false positive rate →
Only detect gross failures (e.g., stuck on zero)

Strategy: under promise, over deliver

- H[real] = lowest expected entropy/bit of source
- H[good] = lowest acceptable entropy/bit of source

- Design source so H[real] > H[good]

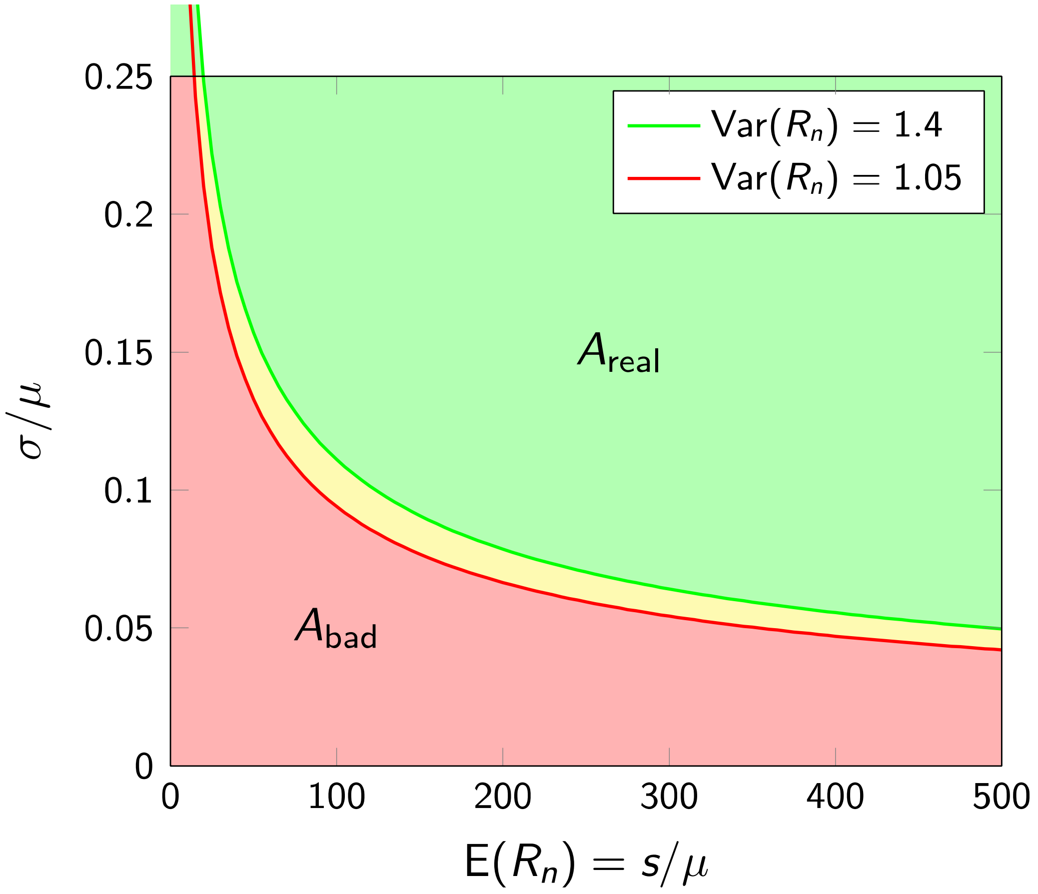
- Health tests detect error when entropy < H[good]



Choosing test parameters

Consider parameter space of entropy source

- A_{real} – where we expect source to reside
- A_{good} – ensures sufficient entropy
- A_{bad} – no guarantee of sufficient entropy



In 90B

- H[submitter] is based on model of source
- Source can claim less entropy than H[submitter]
- Claim entropy/output needed for application
- Health tests tuned to claimed entropy/output of entropy source

Reaction: what happens when test triggers?

failure ➡ signal ➡ detection ➡ reaction

- At some point, raise error condition and stop source

- Some tests fail only when source definitely bad
  - VERY low false positive rate, total failure
  - Shut it down

Generally: very low false positive rate → can't detect much

- Use test with higher false positive rate
- Don't shut down the source on first failed test

Reaction: alternatives to immediate shutdown

- Stop output and run additional tests
- Suppress output until test stops failing
- Don't count outputs as having entropy
- Keep track of failed tests: too many → fatal error
  - E.g., window of last 64 tests run
  - Too many fails → shut it down
- Alter some internal parameters
  - E.g., slow down sampling

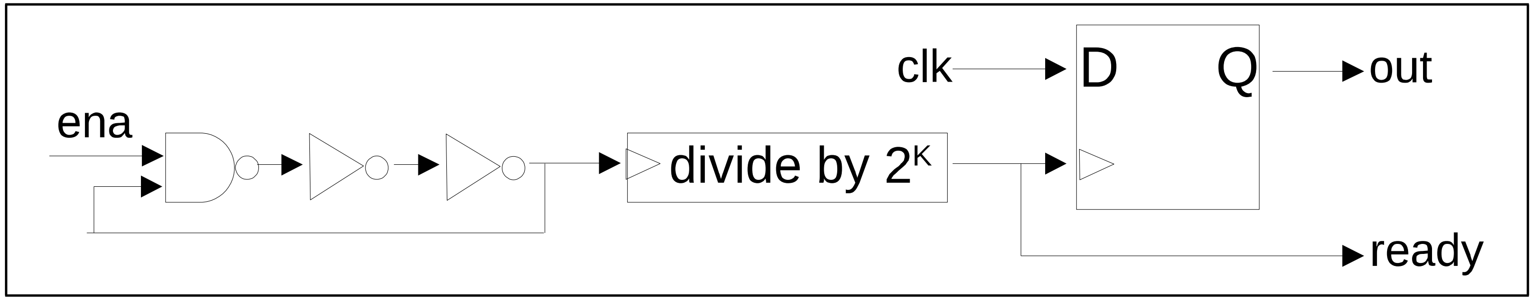
Must be some point at which signal error and stop entropy source

General pattern

failure → signal → detection → reaction

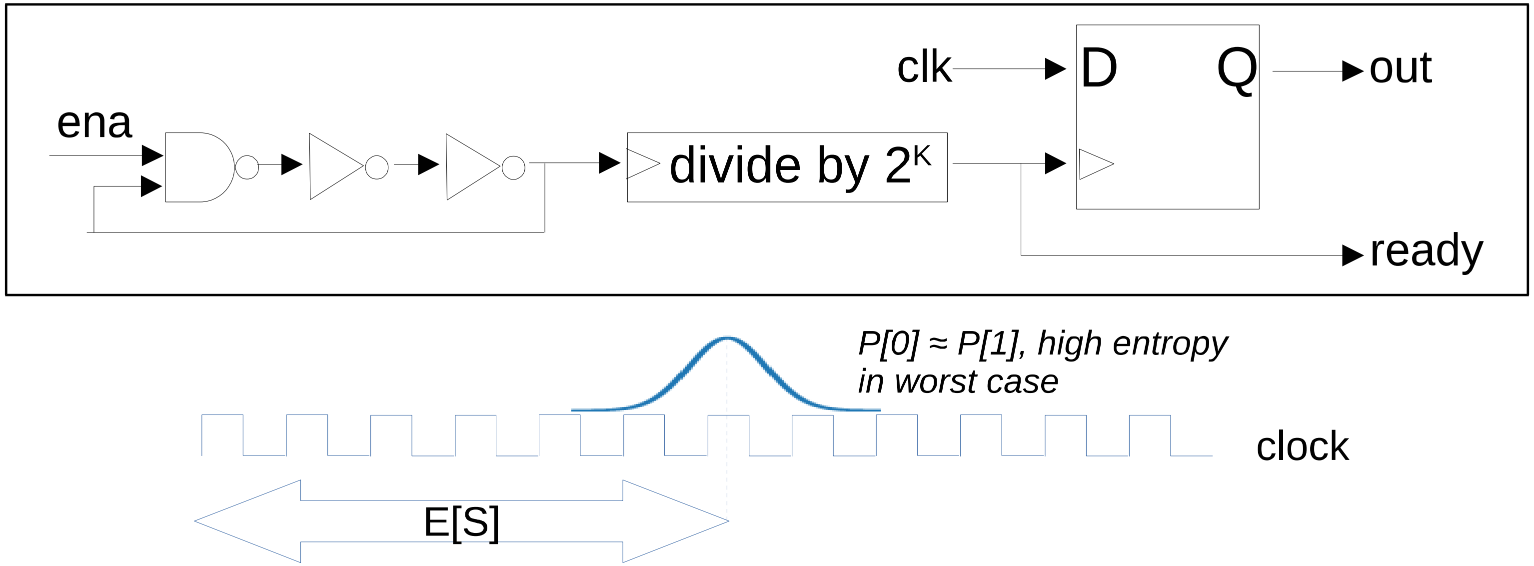
- **Failure:** Something goes wrong causing loss of entropy
- **Signal:** Observable behavior changes
- **Detection:** Health test detects signal
- **Reaction:** Module does something to prevent loss of security

Example 1



- Oscillator locks to clock
- Output bit stuck on constant value
- Repetition count test detects repeated values
- Error condition raised, RNG shuts down

Example 2



- Less variability in sampling interval than expected
  - Ex: σ_{noise} lower than expected
- Measured sampling interval repeats too often
- Health test detects too many repetitions
- Error condition raised, RNG shuts down

Our Continuous Health Tests

- **Repetition Count Test** – Detect when the source gets “stuck” on one output for much longer than expected.

- **Adaptive Proportion Test** – Detect when one value becomes much more common in output than expected.

- Note that tests:
  - Require minimal resources
  - Outputs can be used as they are produced
  - Allow tunable false-positive rates

All we need to know is entropy/sample!

failure → signal → detection → reaction

In our framework

Repetition Count Test – Detect when the source gets “stuck” on one output for much longer than expected.

- **Signal:** too many identical outputs in a row

Adaptive Proportion Test – Detect when one value becomes much more common in output than expected.

- **Signal**: same value appears too many times in a window

Vendor-Defined Tests

- Designers should understand their sources much better than we can.

- Ideally: designers come up with their own health tests, based on
  - How might entropy estimate be wrong?
  - What observable effect will each failure have?

- Our tests are intended as a MINIMUM bar
  - We want vendors to do better.

Vendor-Defined Tests: Requirements

- Submitters need to show that their tests detect the same signals as ours:
  - Detect if a value repeats too often (the source gets stuck).
  - Detect if some value becomes much too likely.

- Submitters can show this by:
  - Proof or convincing argument
  - Statistical simulation

What to test

Raw bits/ raw random numbers

- Often best place to put statistical tests
- Sanity check –can detect catastrophic failures you don't expect

Internal values

- Often only way to detect problem quickly
- Example: Circuit to detect oscillators locking

Conditioned output bits

- Sometimes possible to detect failures through conditioner

Applying test to conditioned outputs

Lots of extra work

Have to show:

- Failure → signal
- Signal shows up in conditioned outputs
- Test detects failure with high probability

Example: testing conditioned outputs

- Failure = oscillator locks to clock
- Signal in raw bits=alternating 10101010… pattern
- Conditioning: Von Neuman unbiasing
- Signal in conditioned bits=same output bit repeated forever
- RCT on conditioned bits will detect that failure

Wrap up

- Health tests critical for entropy sources

failure → signal → detection → reaction

- Failure: Understanding how noise source can fail = first step
  - Conditions that invalidate entropy estimate
- Signal: Observable effect of a failure
- Detection: Test that reliably detect signal
- Reaction: Do something to avoid security loss