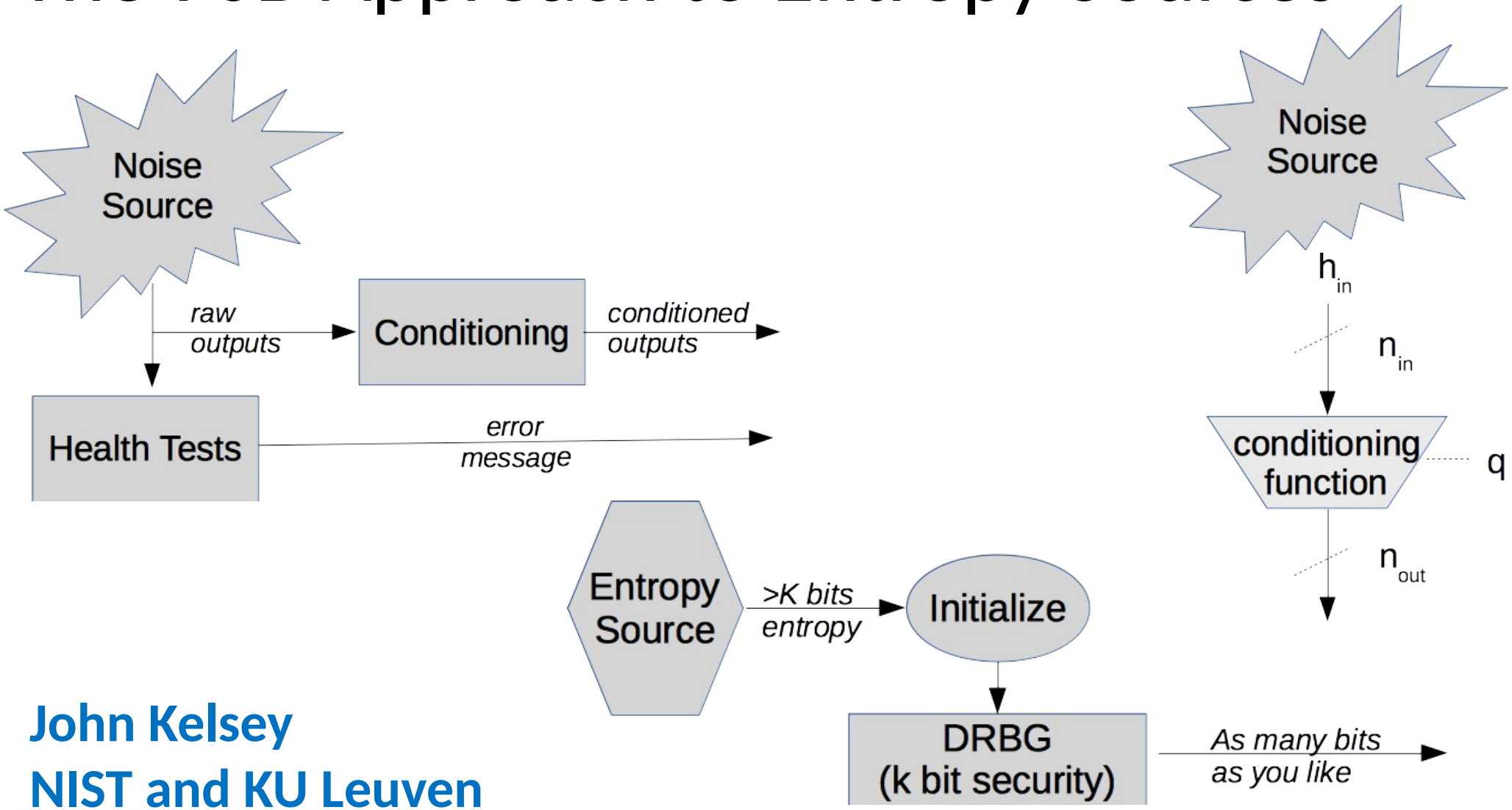
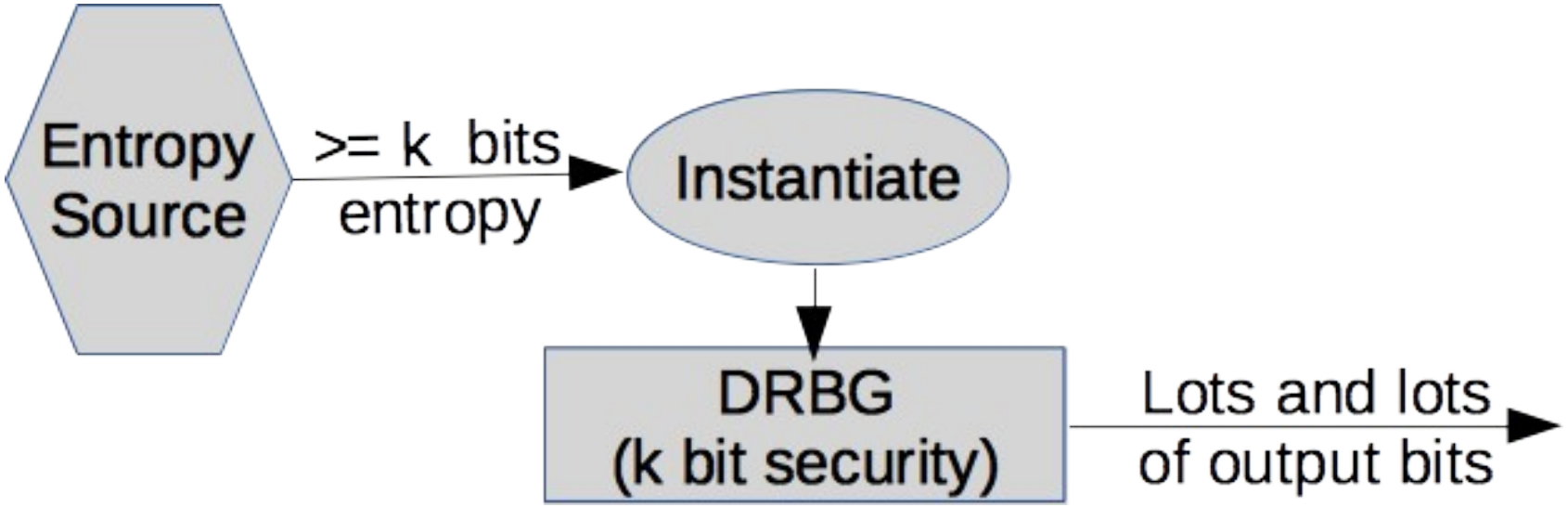
The 90B Approach to Entropy Sources



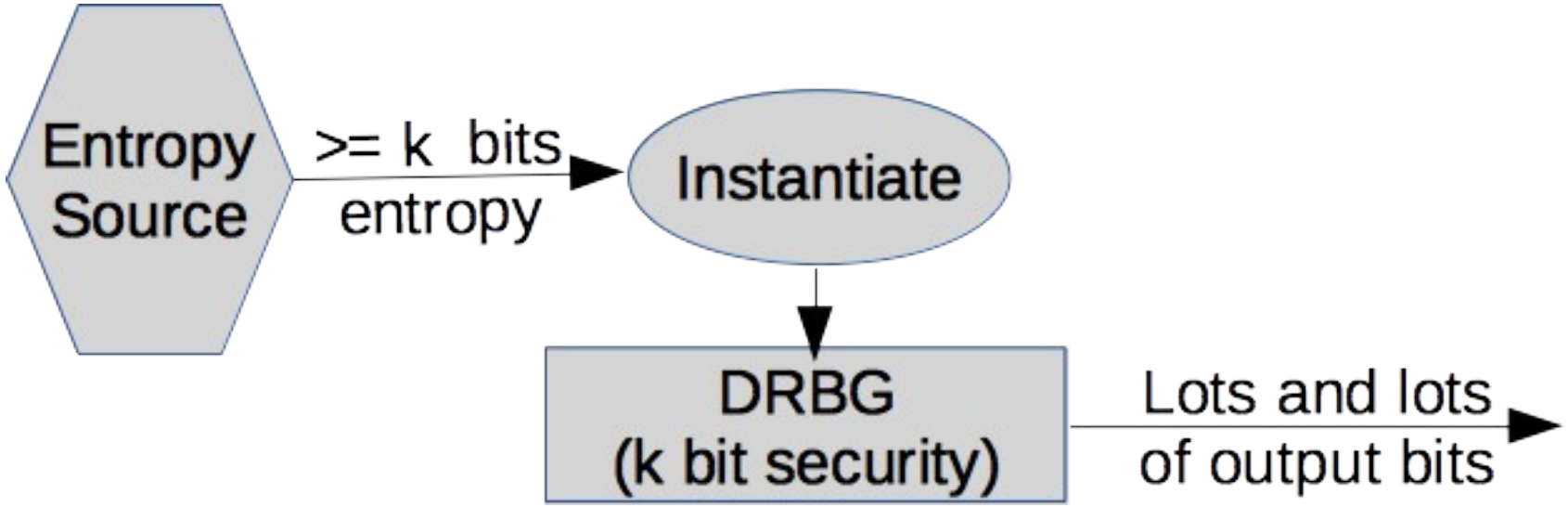
John Kelsey
NIST and KU Leuven

SP 800-90: How Are Random Bits Generated?



- 90A: DRBGs (Deterministic Random Bit Generators)
- **90B: Entropy Sources**
- 90C: Constructing Random Bit Generators (working on it now)

This is the Context for All Our Work on 90B



- DRBG always between entropy source and attackers
- Entropy source must give known amount of entropy
- Not required to give high-quality random bits
  - 0.1 to 8.0 bits of entropy per output

What is an Entropy Source?

- An entropy source provides:
  - Bits or bitstings of fixed length
  - …with a promised min entropy per output
  - …with continuous health testing to ensure the source keeps working
- SP 800-90B is about how to:
  - Build an entropy source
  - Validate an entropy source
- SP 800-90B entropy source != AIS31 entropy source

An entropy source gives you bitstrings, and tells you how much entropy they have.

Components of an Entropy Source

Noise Source

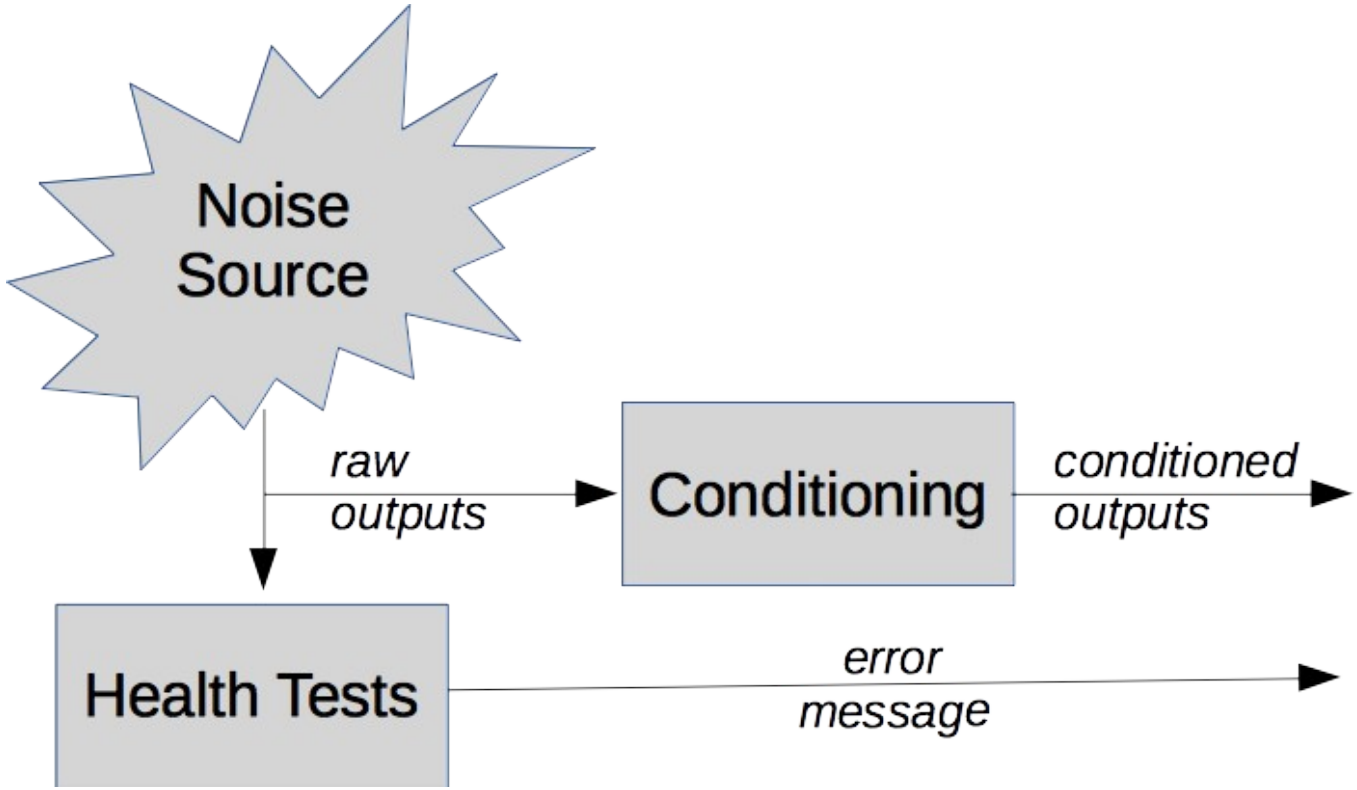
Where the entropy comes from

Health tests

Verify the noise source is still working correctly

Conditioning

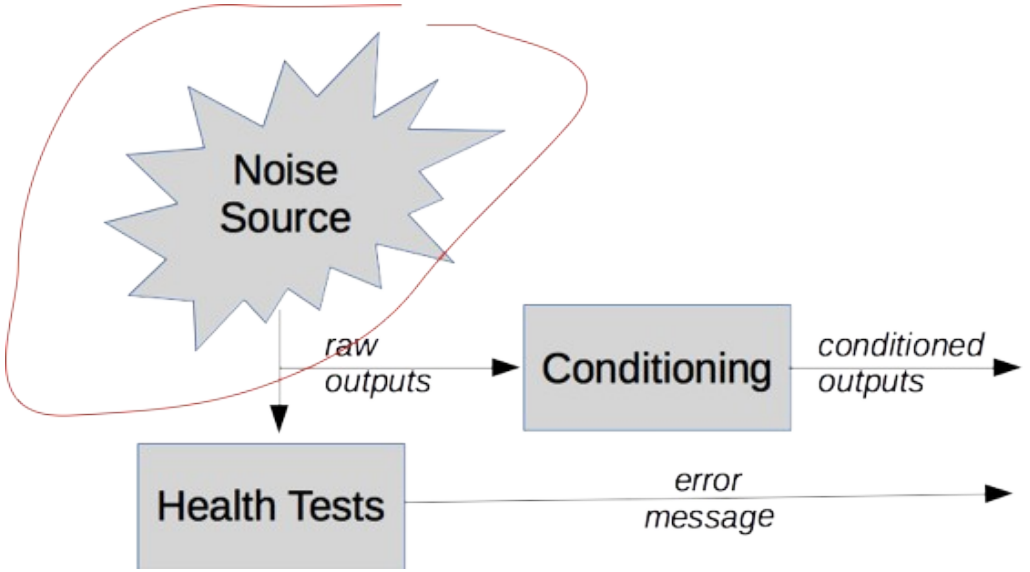
Optional processing of noise source outputs before output.



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

Noise Source

- A noise source provides bitstrings with some inherent unpredictability.
- Nondeterministic mechanism + sampling + digitization = noise source.
- **Every entropy source must have a noise source.**

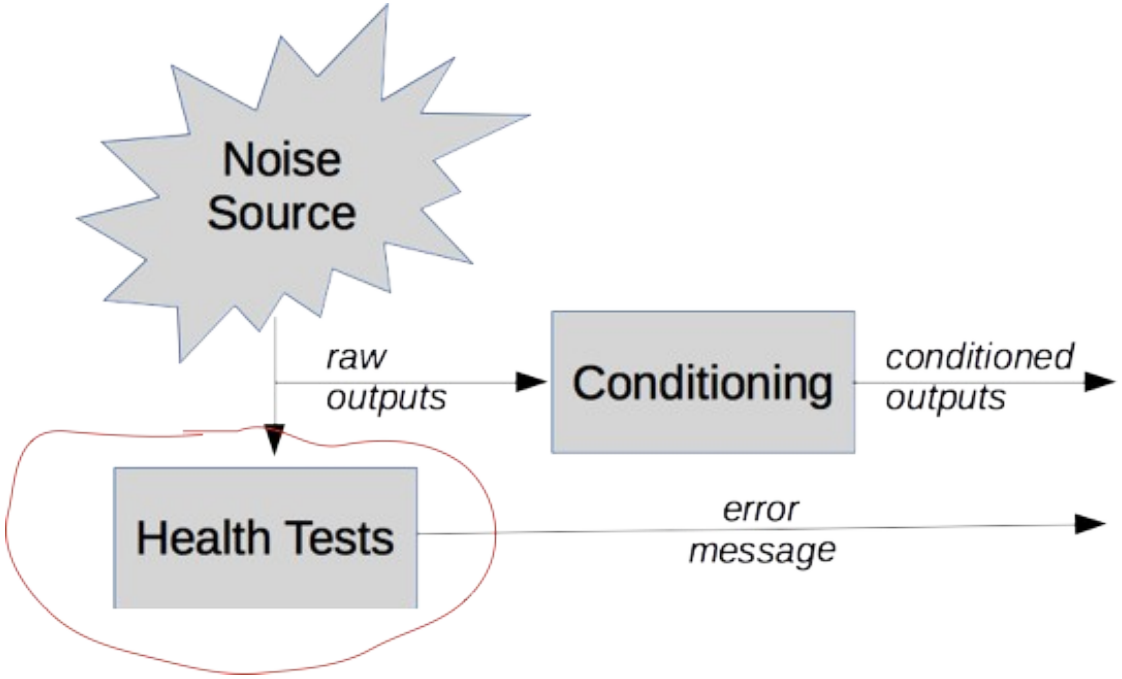


Submitter:

- Explains how it works in detail
- Provides an entropy estimate
- Provides a justification for entropy estimate

- **The noise source is the core of an entropy source**—it's the thing that actually provides the unpredictability

Health Tests

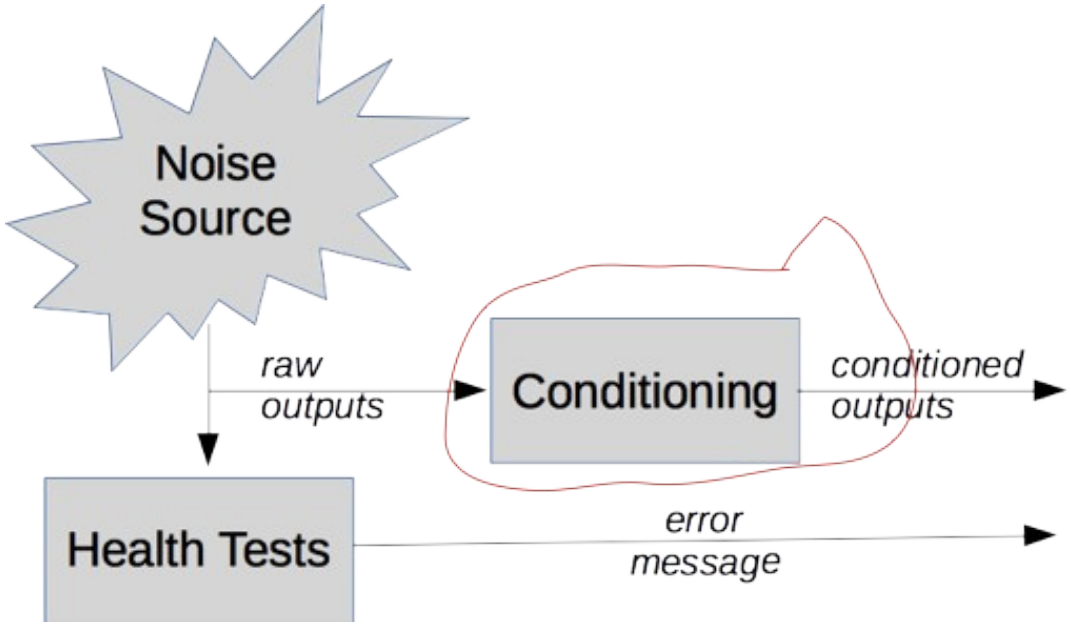


- Goal: detect **major failures**
- **REQUIRED** for all entropy sources.

- Three flavors:
  - Startup testing = one-time test done before any outputs are used
  - Continuous testing = testing done in the background during normal operation
  - On Demand testing = testing when requested by application

Conditioning

- Process noise source outputs to improve entropy/bit.
- Deterministic (can’t add entropy)
- **Conditioning is optional**
- Can be cryptographic (SHA256)
- Or not (Von Neumann unbiasing)

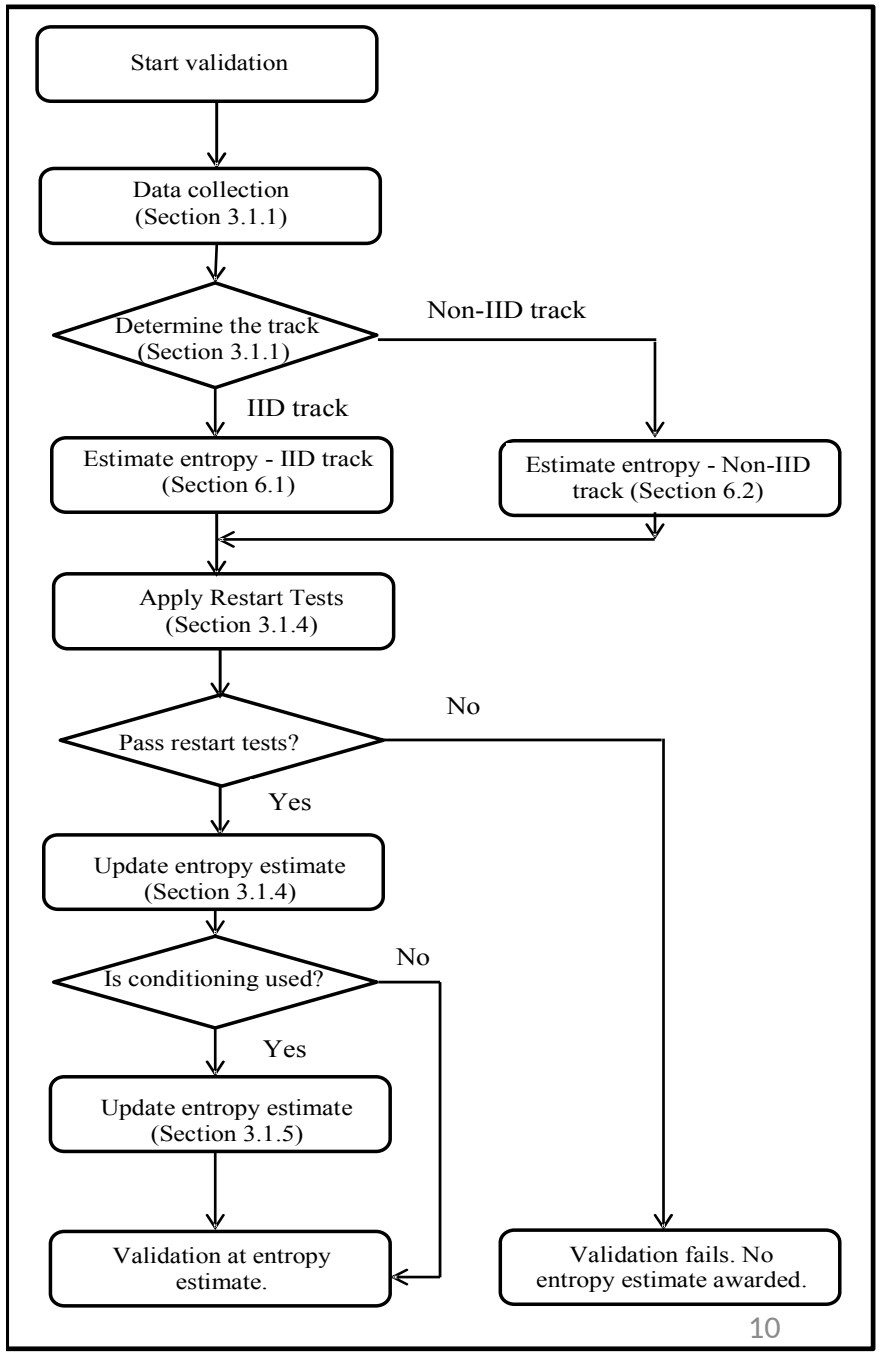


90B Requirements for These Components

- Noise Source
  - Documentation and description
  - Entropy estimate and justification
  - **Entropy assessment via testing**
- Health Tests
  - **Using our tests OR**
  - **Designing custom tests and justifying them**
- Conditioning
  - **Choice of conditioning functions**
  - Documentation
  - **Entropy accounting**

Entropy Estimation: Testing the Noise Source

- Submission package
- Data collection
- IID vs non-IID
- Entropy estimation
- Restart tests



Submission Package

- **Detailed description of source**
  - Noise source
  - Health tests
  - Conditioning
- Entropy estimate from noise source
  - With justification and analysis
- Claim of IID or non-IID source
- Data collected by lab or by vendor

What Data is Collected?

Raw Noise Source:

- Sequential dataset: 1,000,000 successive samples from noise source
- Restart dataset: 1,000 restarts, 1,000 samples from each restart
  - Restart = power cycle, hard reset, reboot

Conditioner Outputs—sometimes required*

- Conditioner dataset: 1,000,000 successive samples from conditioner
- *Only for non-vetted conditioning function*

* *Not used in entropy estimation*

Determining the Track (IID or Non-IID)

IID = Independent and Identically Distributed

- IID means each sample independent of all others, independent of position in sequence of samples

How do we determine if source is IID?

- ONLY if designer claims source is IID (with rationale)
- Run statistical tests to try to disprove claim.
- If we can’t disprove it, we assume source is IID for entropy estimation

IID Tests

- Main idea of IID tests: Permutation
- Compute a set of statistics on original dataset
- For j = 1 to 10000:
  - Randomly permute original dataset.
  - Compute set of statistics on permuted dataset.
- If any original statistic is in the top or bottom five values
  - .9995 or .0005
- then reject the claim that the source is IID.

- Also do chi-square tests of independence and goodness of fit.

Entropy Estimation

IID Case:

- Count most common value in output and construct bound on P_{max}.

Non-IID Case:

- Apply many different entropy estimators against sequential dataset.
  - Parameter estimation tests (NSA)
  - Predictor tests (NIST)
  - Longest repeated substring + K-tuple estimate (NIST)
- Take minimum of all estimates.

Combining Estimates

- H[original]
  - We do entropy estimation (iid or noniid) on the sequential dataset.
- H[binary]
  - If the sequential dataset is not binary, we convert it to binary and do a second estimate on the first 1,000,000 bits that result.
- H[submitter]
  - The submitter had to estimate the entropy in his documentation package.
- H[I] = min(H[original,H[binary]*N,H[submitter])
  - Just take the smallest estimate of the three (or two, if it's binary data)

* Final entropy estimate is **never** larger than submitter estimate!

Restart Tests: Rationale

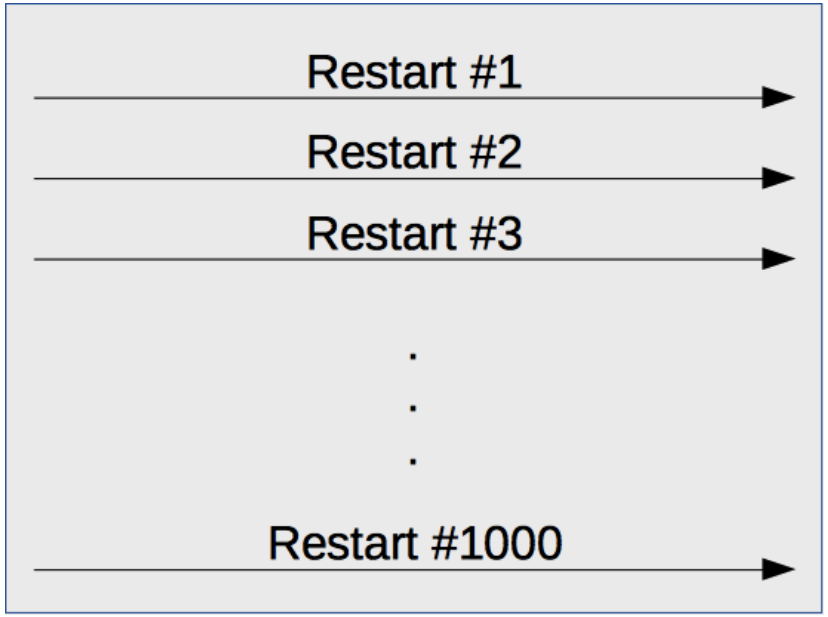
- Some noise sources look a lot worse *across restarts*.
  - Restart = power cycle, hard reset, reboot
- Goal: Detect predictability that only becomes apparent when examining many sequences generated by a source across restarts.

Good noise source should be:

- Same across restarts
- Same distribution at all positions in output sequence

If these aren't true, we want to reject the source.

Restart Dataset



Restart dataset is a 1,000 x 1,000 matrix of samples

- Each row is 1,000 successive samples, taken after the source restarted
- Column K is the Kth sample from each restart.

Sanity Check: Row and Column Binomals*

- Start with entropy estimate H[I] from previous step.
- Use to compute P[max] = maximum prob of any value
- Use P[max] to compute upper bound (C)
  - # times most common value should appear in any 1000 samples.
- If any value appears more than C times….
  - in any column
  - or any row
- …then the source fails.

* There was an error in the formula in version 1 of 90B, being fixed now.

Row and Column Datasets

- Row dataset = all rows of matrix, concatenated
  - 1000 samples from 1st restart || 1000 samples from 2nd restart || …
- Column dataset = all columns of matrix, concatenated
  - All 1st samples after restart || all 2nd samples after restart || …
- Run entropy estimators on row and column dataset
- H[r] = row dataset estimate
- H[c] = column dataset estimate

* IID sources are handled a bit differently!

Using the Row and Column Estimates

- If

  H[r] <= 0.5 H[I]

- or

  H[c] <= 0.5 H[I]

- Then reject the source*.
- Otherwise:
- H[final] = min(H[r], H[c], H[I])

* This is a kind-of arbitrary line, but this big a drop in entropy suggests a major problem.

Entropy Estimation Wrapup

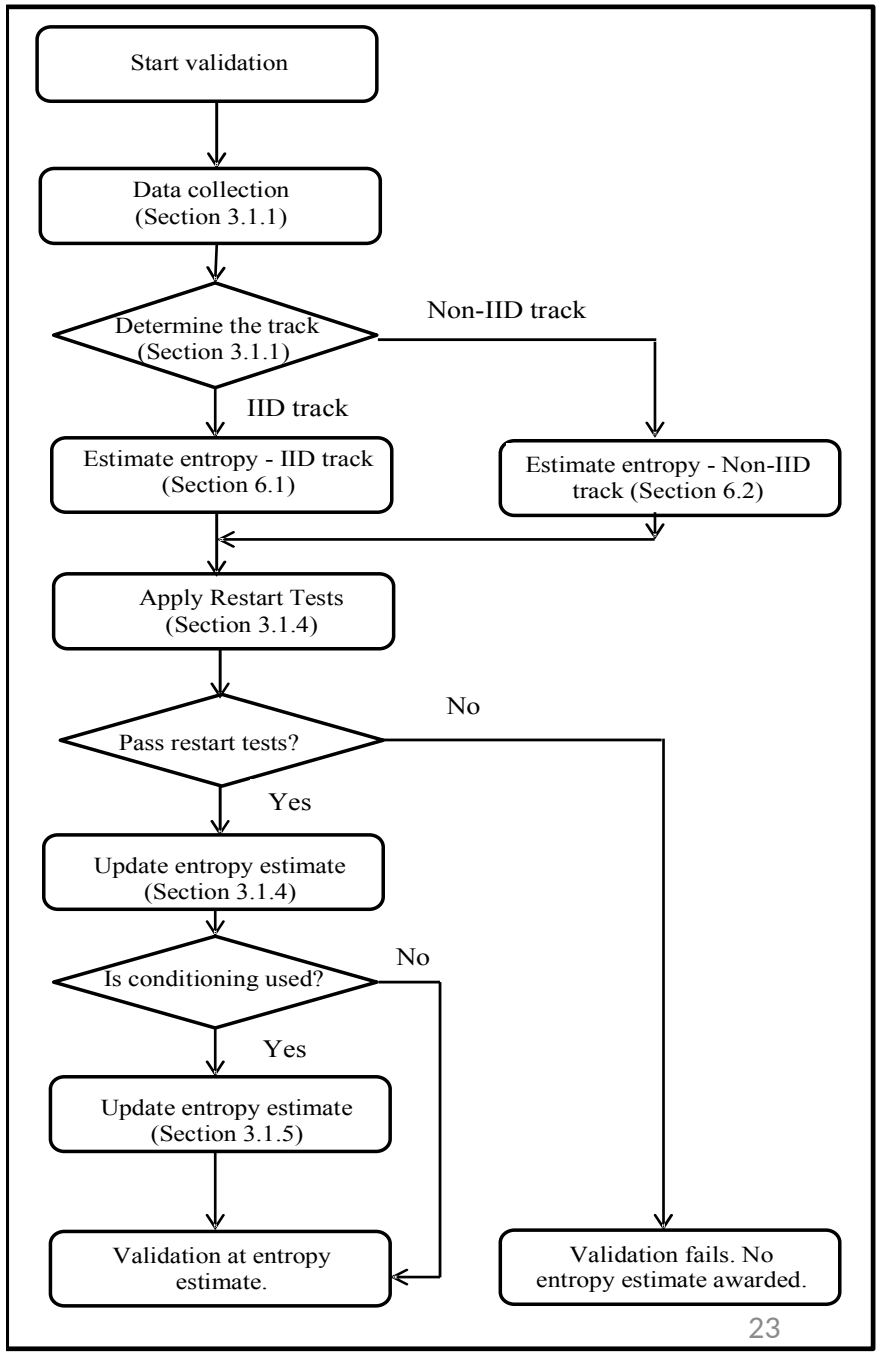
- H[original] = original entropy estimate from sequential dataset
- H[submitter] = submitter's original entropy estimate
- H[binary] = estimate from binary version of data (sometimes)
- *H[r]* = row dataset entropy estimate
- *H[c]* = column dataset entropy estimate

- All combined to give us the final estimate:

  H[final] = min(H[original],H[submitter],H[binary]*N,*H[r]*,*H[c]*)

Entropy Estimation: Wrapup

- Submission package
- Collect data
- IID or non-IID track?
- Estimate entropy on sequential data
- Restart testing
  - May fail the source
  - May lower the estimate
- h[final] = minimum of all estimates
  - sequential, submitter, binary, row, column,
- Either FAIL source, or we have entropy estimate **for noise source**.



So, What's the Problem With All This?

- No set of general-purpose statistical tests can measure the entropy per sample in an arbitrary sequence of values.
- Right way to build a noise source is:
  - Design your noise source
  - Understand it
  - Model it
  - Use your model to estimate its entropy
  - Run general-purpose tests on outputs as a sanity check.
- We require design documentation and an entropy estimate from designer to support this…
- …but we're limited in what resources we can demand for validation testing, and what expertise we can require from labs.

This approach is the best we know how to do given existing constraints.

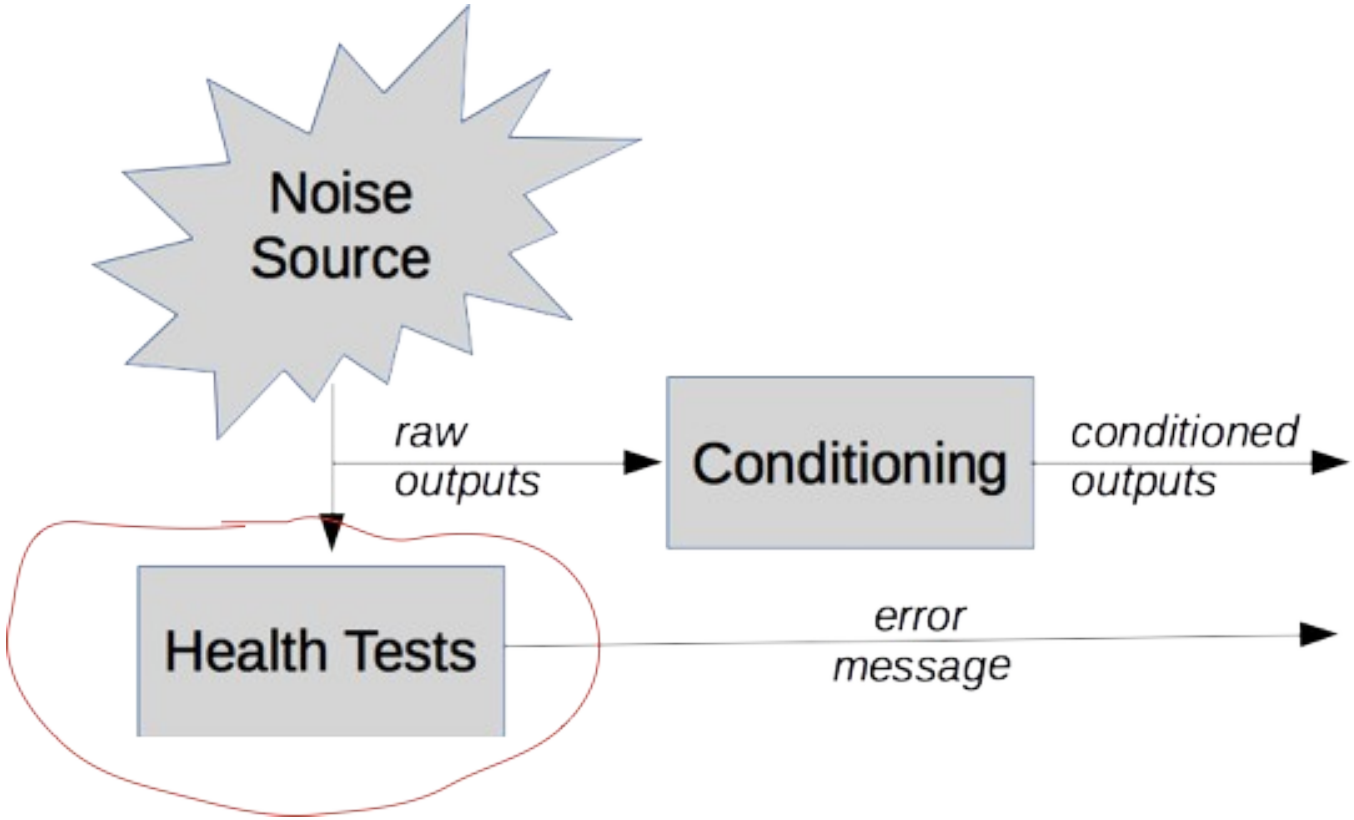
Health Testing

Noise sources are fragile.

Health testing should detect major failures.

Three categories:

- Continuous
- Startup
- On Demand



Health Tests: Big Picture

Noise sources can be fragile. Thus, all entropy sources MUST have health tests.

- Health tests examine the **raw noise source outputs**, NOT conditioned outputs.

- Tests must have minimal performance impact.
  - False positive rate is important
  - Almost no memory
  - Almost no computation per sample

- 90B has two recommended continuous tests
  - Use ours or make up your own and show they’re okay.

Health Tests: Continuous / Startup / On Demand

- Continuous Tests
  - Going on all the time behind the scenes
  - Minimal resources
  - No buffering required

- Startup Tests
  - Run on a sequence of noise source samples at startup, before those samples may be used for anything.
  - If the tests detect a problem, then the device refuses to function.

- On Demand
  - Run when requested
  - Might just be rerun of the startup tests

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!

Vendor-Defined Tests

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

- Ideally: designers come up with their own health tests, based on
  - Understanding of likely failure modes of source.
  - Probability model used to derive entropy estimate.

- 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 accomplish the same broad goals 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 convining argument
  - Statistical simulation

Startup Testing

- FIPS 140 requirement

MINIMUM REQUIREMENT:

- The noise source generates a pool of at least 1024 samples, with the continuous health tests running. The samples CANNOT be used until the testing is complete.
- If no error is detected, then the module MAY use or discard those samples.

The designers MAY include additional tests at startup.

- …and SHOULD!

Our requirements are a minimum bar—a good design should do better!

On Demand Testing

- On demand testing happens when the application requests it.

MINIMUM REQUIREMENT:

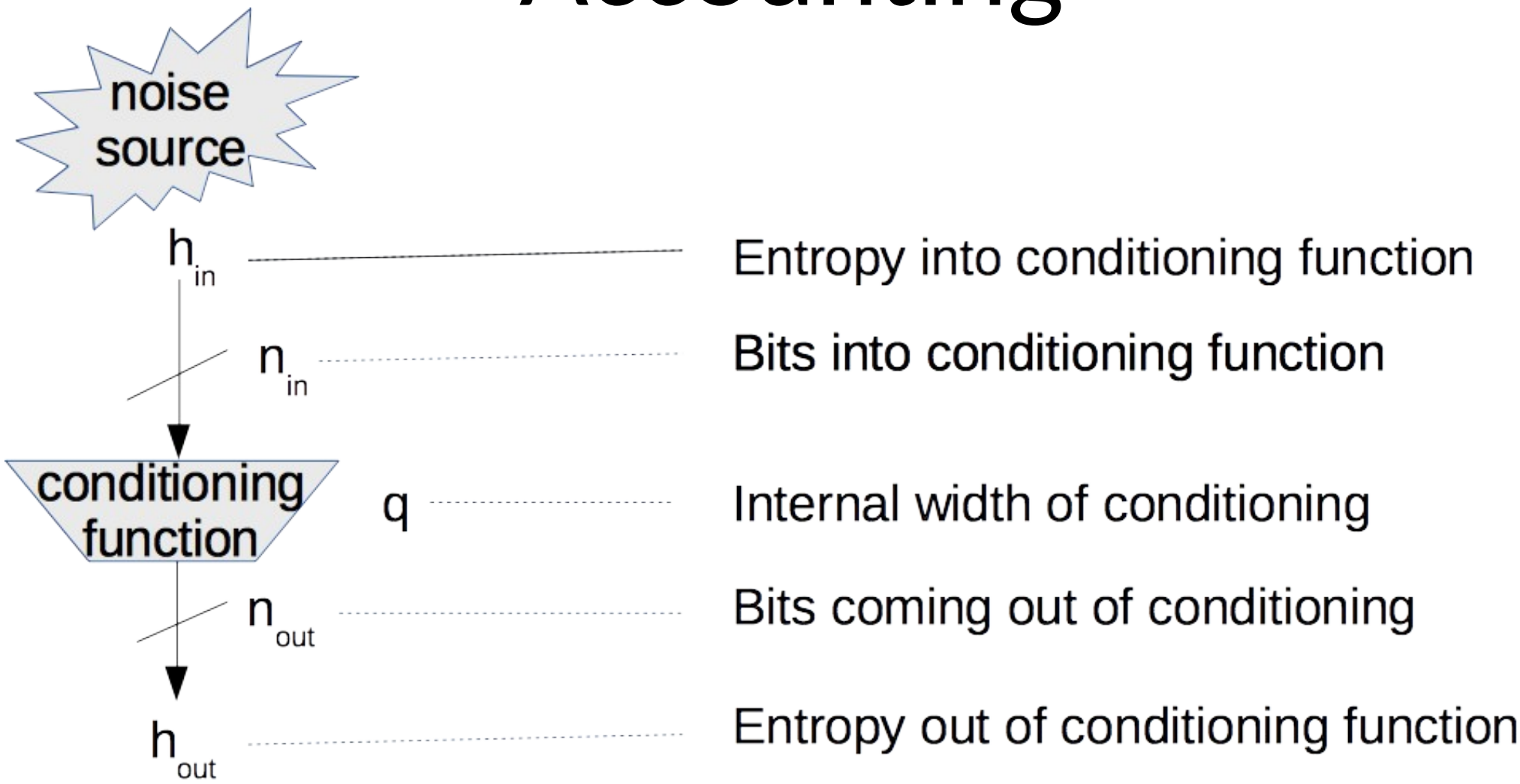
- Rerun the startup test.

The designers MAY include additional tests for on-demand testing…

- …and SHOULD!

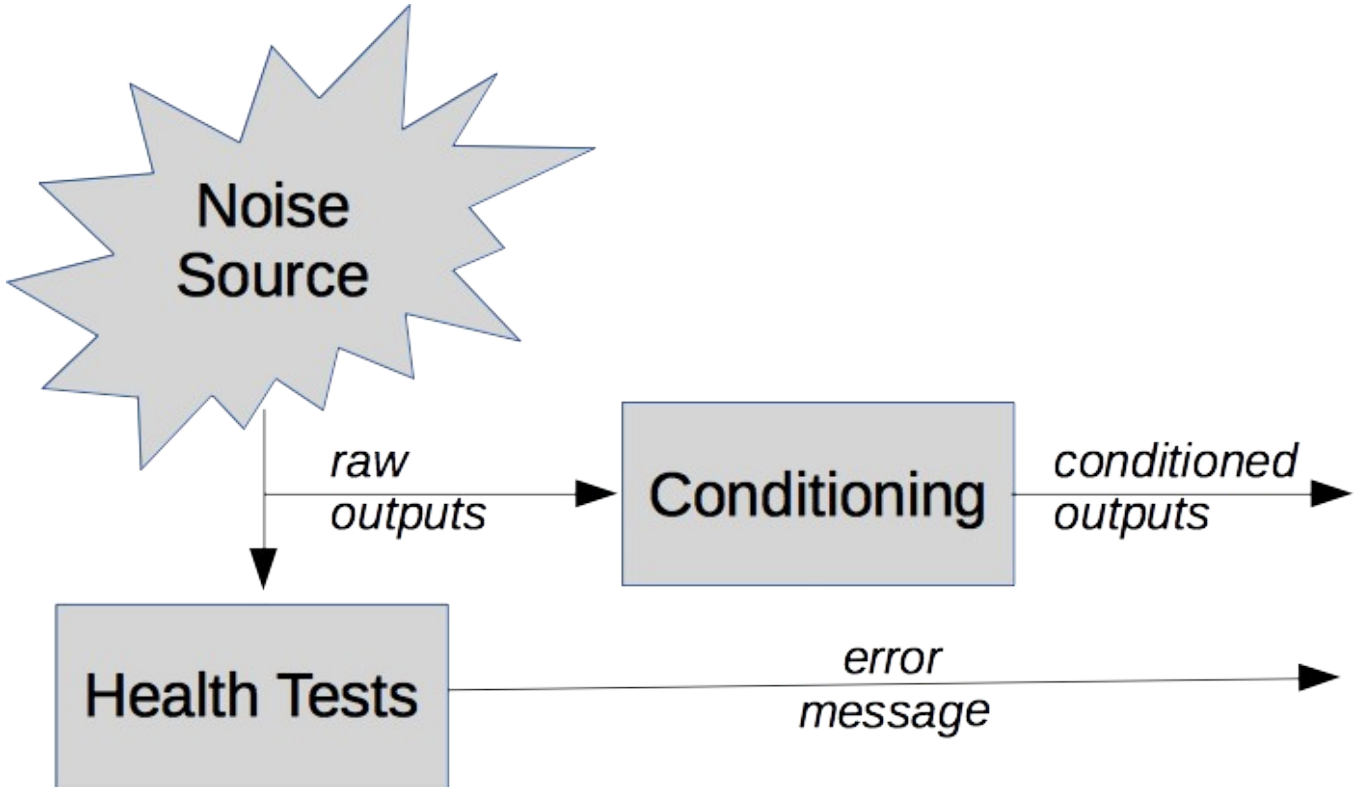
Again, these requirements are a minimum bar.

Conditioning and Entropy Accounting



Conditioning

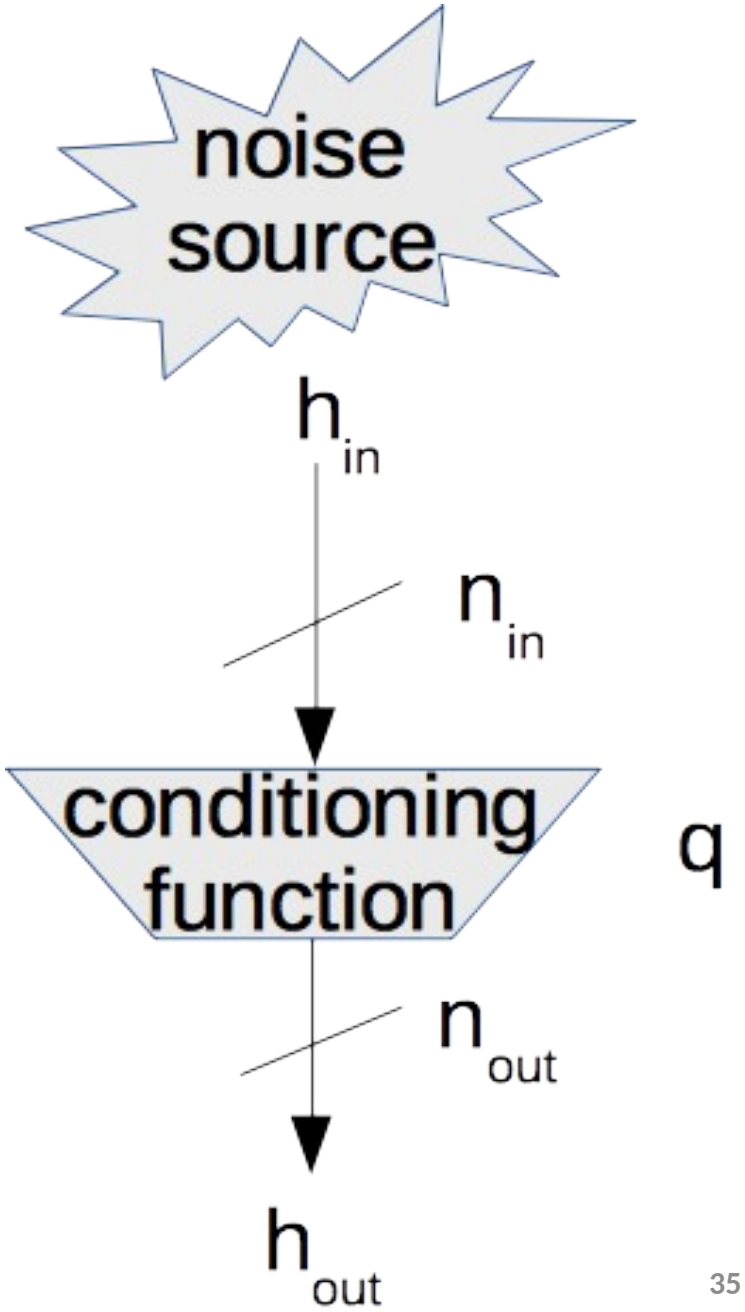
- Optional—not all entropy sources have it.
- Improve statistics of outputs
- Some conditioners can allow the source to produce full-entropy outputs.



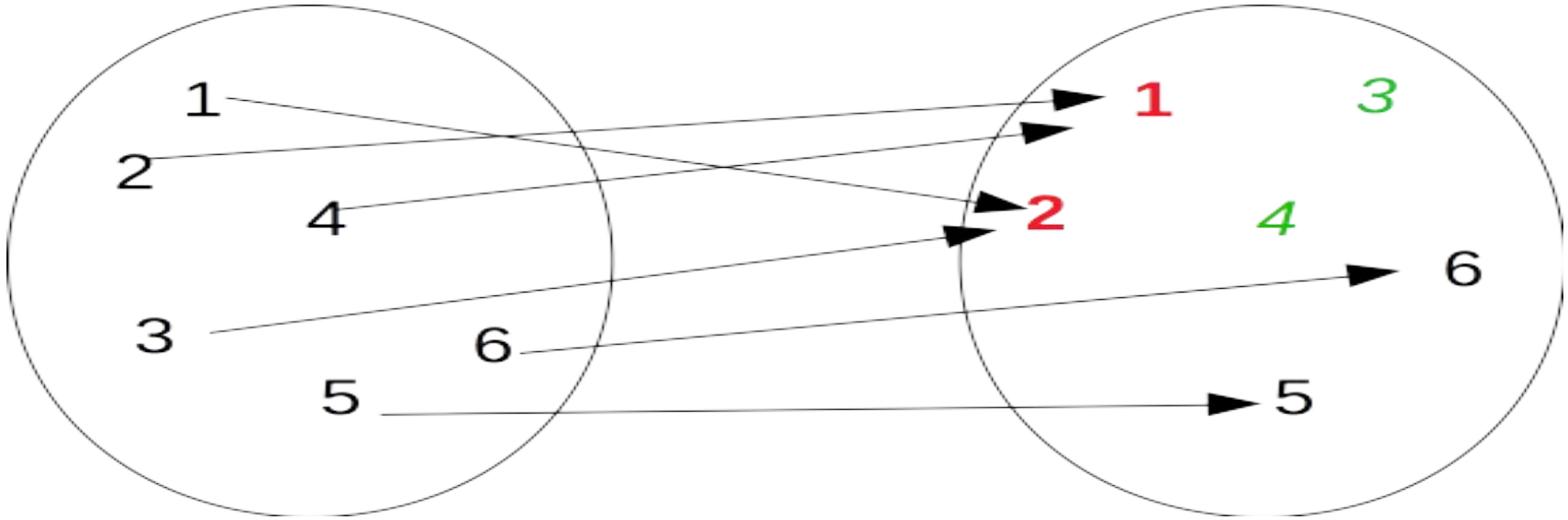
The Big Picture

- Noise source: h_{in} bits entropy.
- Conditioner:
  - Takes n_{in} bits of input
  - Yields n_{out} bits of output.
- How much entropy do we get per output? $\boldsymbol{h_{out}}$

Figuring out h_{out} is the whole problem.



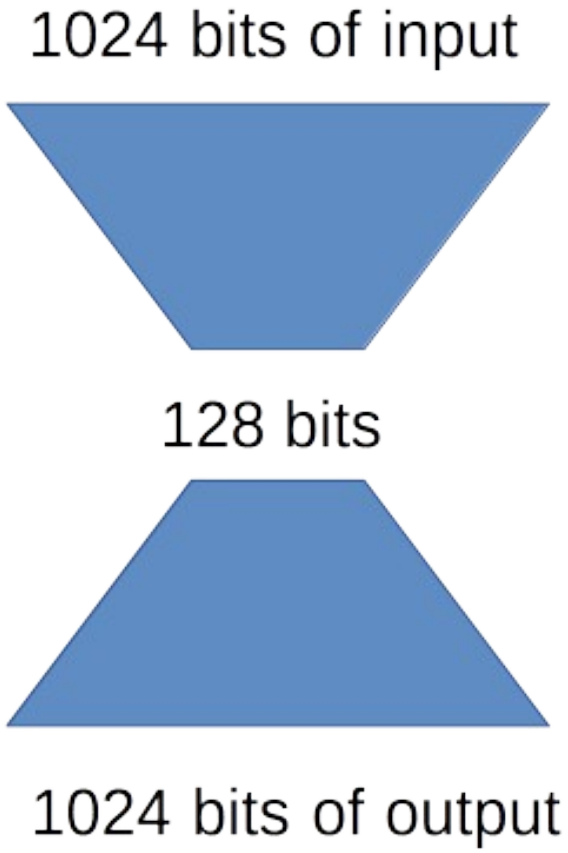
Internal Collisions



- Suppose we have a random function F() over n bits.
- If we feed it 2^n different inputs, do we expect n bits of entropy out?
- NO! Because of *internal collisions*.
  - Some pairs of inputs map to the same output
  - Some outputs have no inputs mapping to them
- Internal collisions have a big impact on how we do entropy accounting!

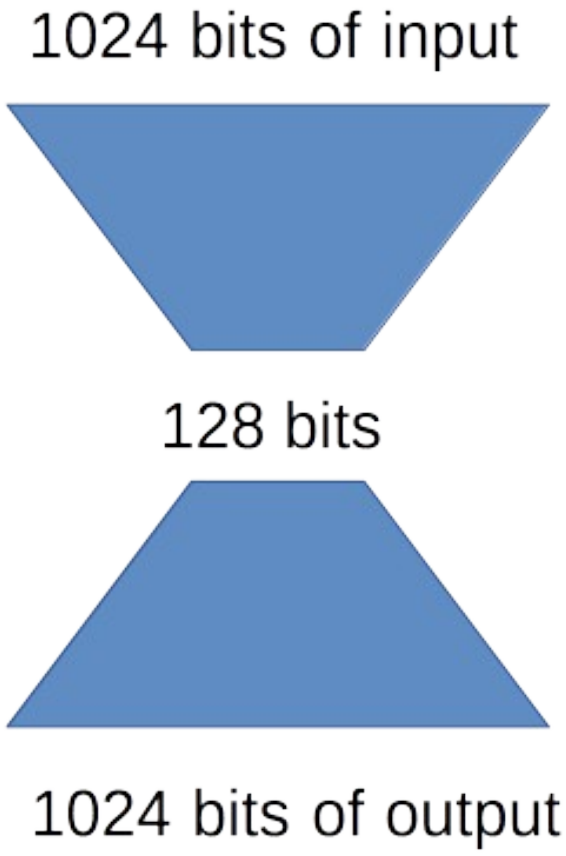
Internal Width of a Function

- Imagine a function that:
  - Takes a 1024 bit input
  - Maps it down to a 128-bit internal state
  - Generates a new 1024-bit output from that state
- It's obvious that this function can't get more than 128 bits of entropy into its output.
- This is the idea behind *internal width* (q) of a function
- In this case, q = 128
- q is the "narrow pipe" through which all entropy must pass.



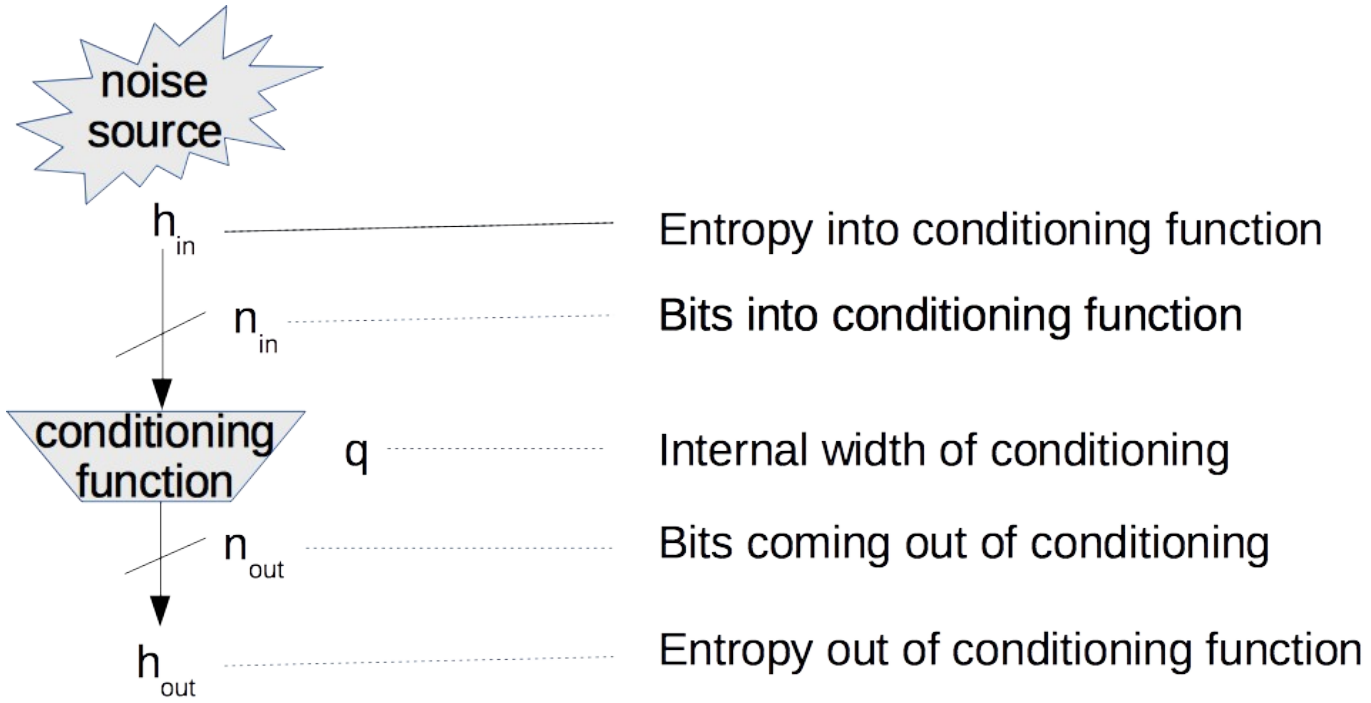
Relevance of Internal Width

- No matter how much entropy goes into the input of this function, no more than 128 bits can ever come out…
- …because the output is entirely a function of those 128 bits of internal state.
- Our formulas for entropy accounting consider the **minimum** of output and internal width.
- Internal collisions apply just as much to internal width as to output size.



$\mathbf{Output\_Entropy}(n_{in}, n_{out}, q, h_{in})$:

1. Let and $P_{high} = 2^{-h_{in}}$ and $P_{low} = \frac{(1-P_{high})}{2^{n_{in}-1}}$.
2. $n = \min(n_{out}, q)$.
3. $\psi = 2^{n_{in}-n} P_{low} + P_{high}$
4. $U = 2^{n_{in}-n} + \sqrt{2\, n(2^{n_{in}-n})\ln(2)}$
5. $\omega = U \times P_{low}$
6. Return $-\log(\max(\psi, \omega))$

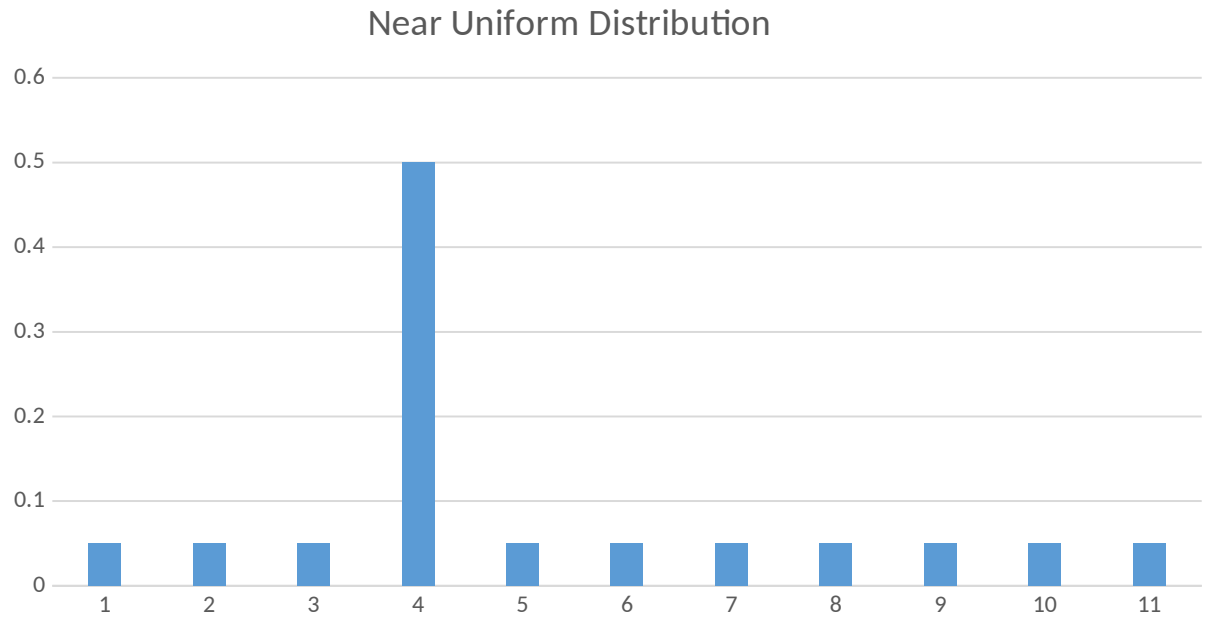


*Note change from draft version!

*Note change from draft version!

Simplifying Assumptions*

- Given min-entropy: h_{in}
- Assume **near-uniform** dist:
  - $P_{high} = 2^{-h_{in}}$
  - $P_{low} = \frac{(1-P_{high})}{2^{n_{in}-1}}$
- Conditioner maps input → output randomly



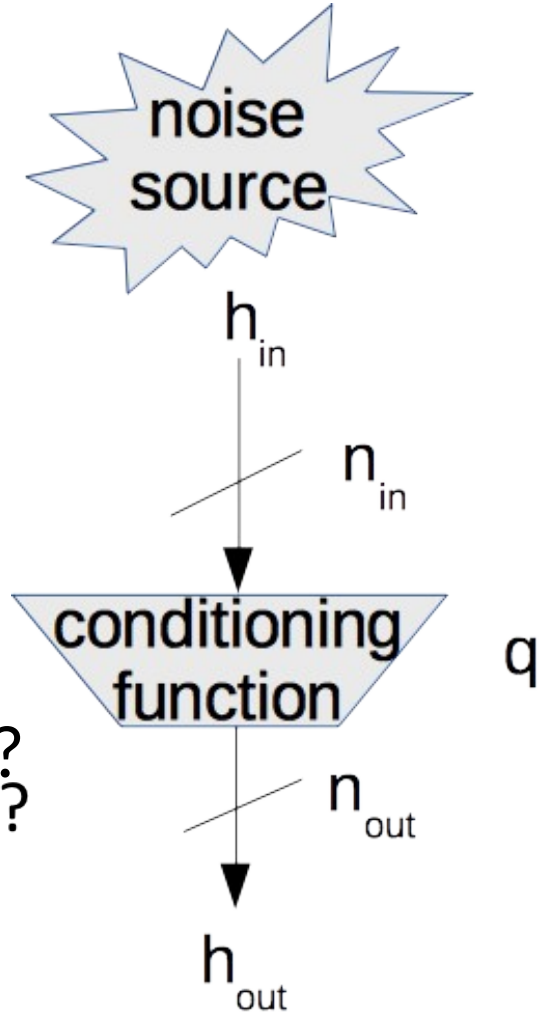
*Simplifying assumptions are necessary to make this workable.

*Simplifying assumptions are necessary to make this workable.

We Need Prob of MOST PROBABLE output!

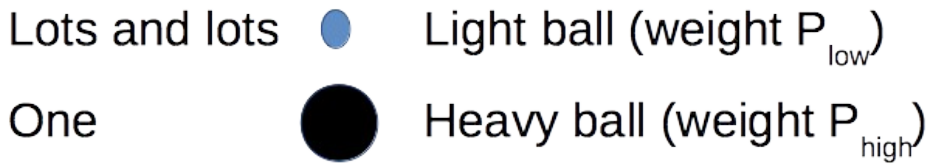
- Each input mapped to a randomly-selected output
- We have one high prob input ($P_{high} = 2^{-h_{in}}$)
- We have $2^{n_{in}}$ low prob inputs ($P_{low} = \frac{(1-P_{high})}{2^{n_{in}-1}}$)
- We have $n = \min(n_{out}, q)$ bins.

- Question: What's probability of MOST probable output?

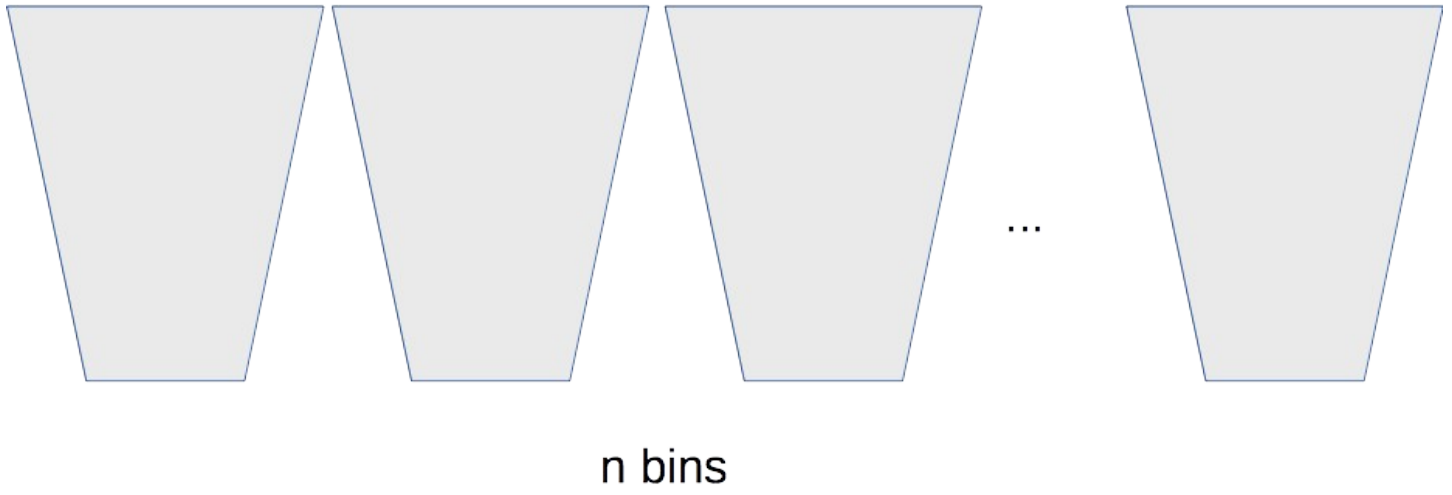


Analogy: Tossing Balls into Bins

- Imagine we're tossing balls into bins.



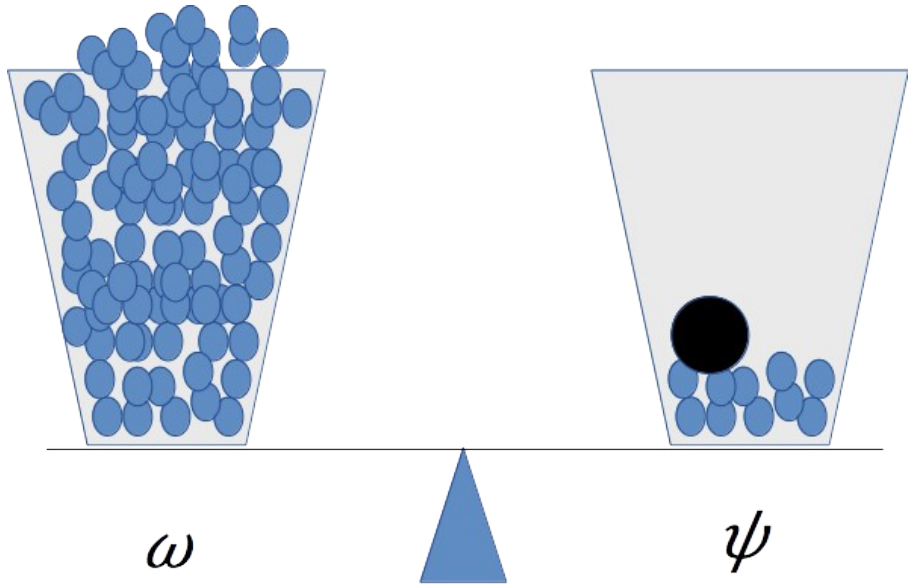
- One heavy ball
- …lots of light balls.



- When we're done, what's the heaviest bin?

Tossing Balls into Bins….

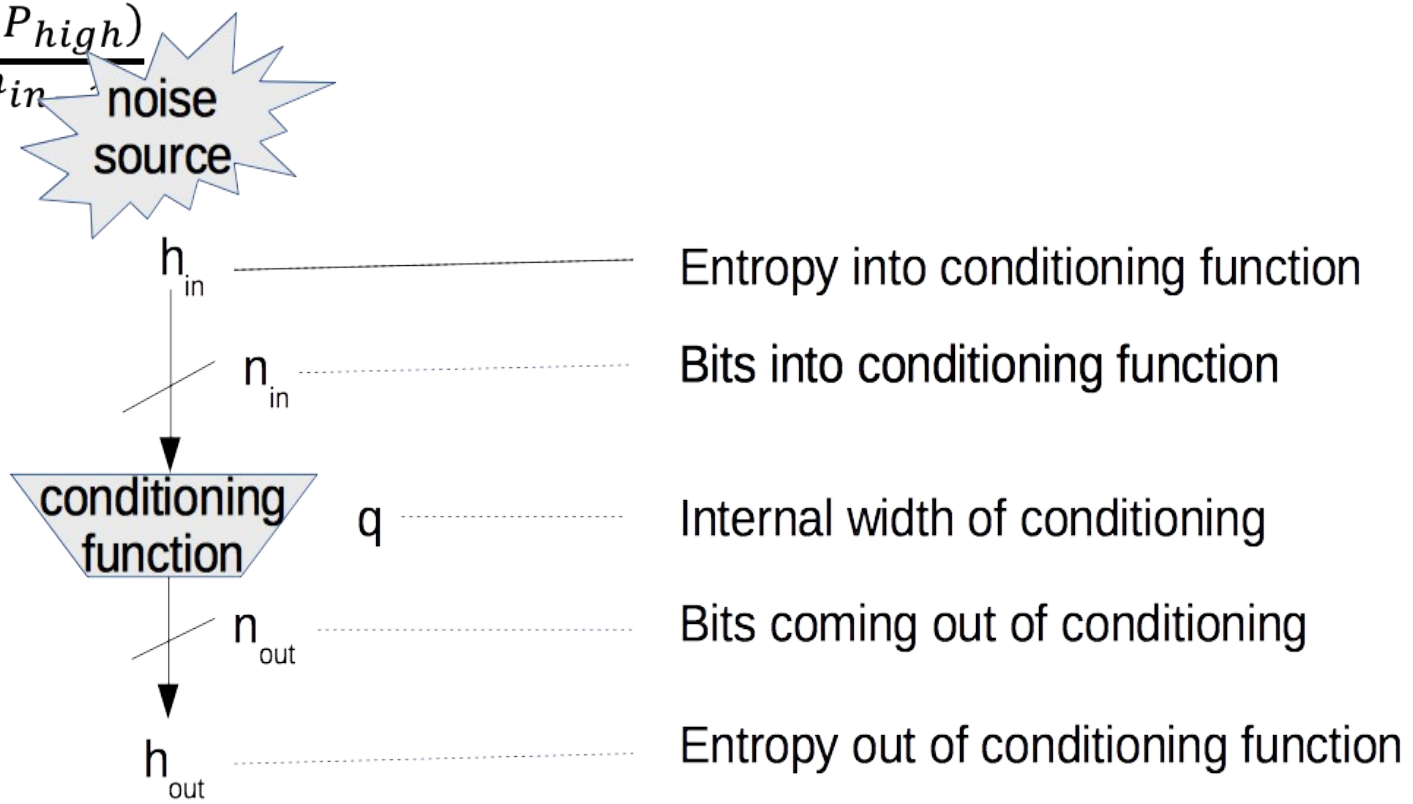
Which bin is the heaviest?



- Two possibilities:
  1. Bin into which the heavy ball falls (ψ)
     - Weight of average number of light balls + weight of heavy ball
     - $n_{in} - n$ of average number of light balls + weight of heavy ball
     - $\psi = n_{low} + 1_{high}$
  2. Bin into which the largest # of light balls fall (ω)
     - U = # of balls in bin with the most balls.
     - $U = 2^{n_{in}-n} + \sqrt{2\, n(2^{n_{in}-n})\ln(2)}$ (Raab and Steeger, 1998)
     - $\omega = U \times P_{low}$
     - U = (Raab and Steeger, 1998)

$\mathbf{Output\_Entropy}(n_{in}, n_{out}, q, h_{in})$:

1. Let and $P_{high} = 2^{-h_{in}}$ and $P_{low} = \frac{(1-P_{high})}{2^{n_{in}}}$
2. $n = \min(n_{out}, q)$.
3. $\psi = 2^{n_{in}-n} P_{low} + P_{high}$
4. $U = 2^{n_{in}-n} + \sqrt{2\, n(2^{n_{in}-n})\ln(2)}$
5. $\omega = U \times P_{low}$
6. Return $-\log(\max(\psi, \omega))$

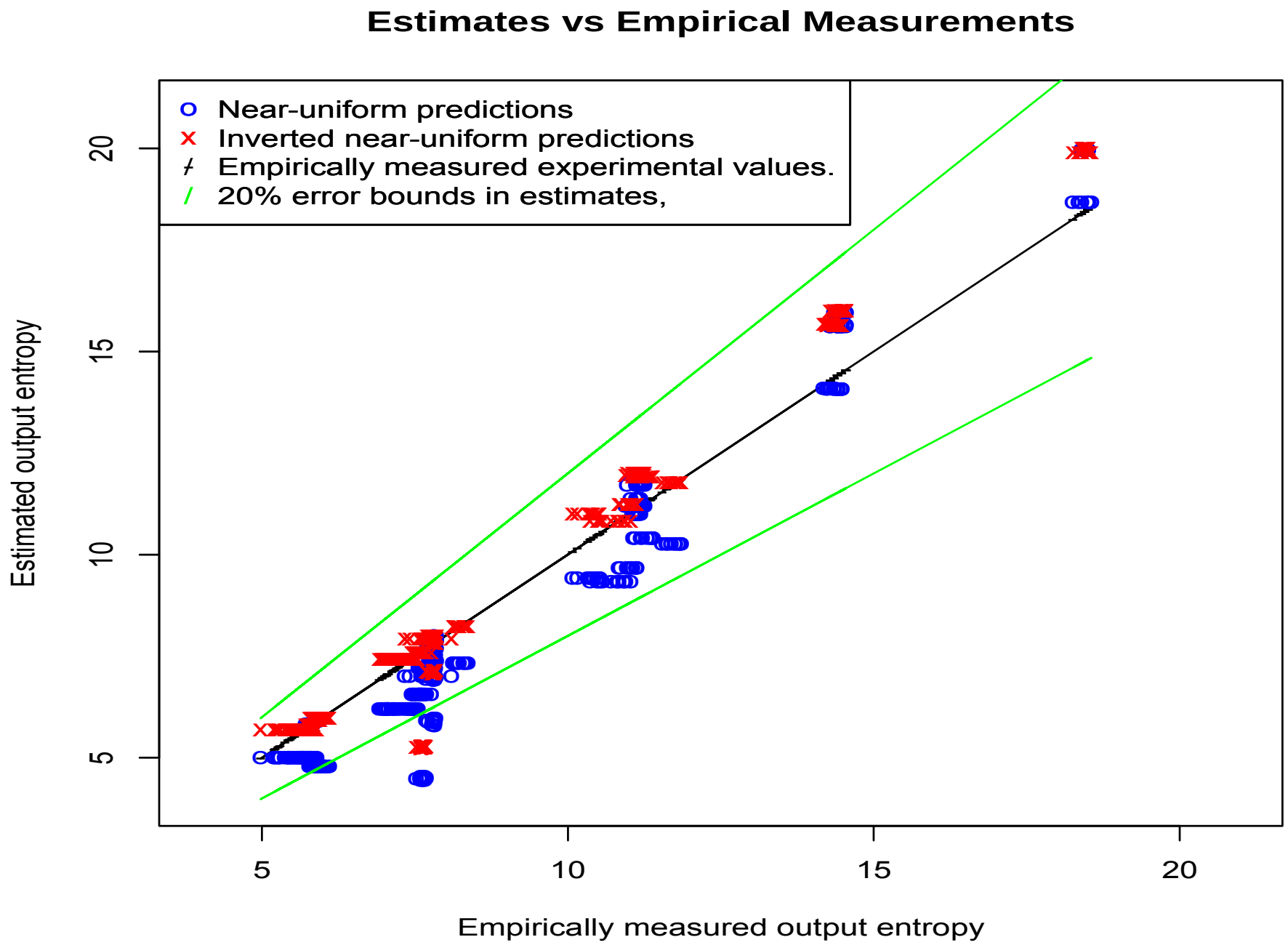


*Line 4 adapted from formulas provided in [RaSt98].

*Line 4 adapted from formulas provided in [RaSt98].

Checking the assumptions

- We did a lot of simulation of small cases
- Many distributions
- …including non-IID ones
- With good (cryptographic) conditioners, the formula describes reality very well.


Estimates vs Empirical Measurements
Near-uniform predictions
Inverted near-uniform predictions
Empirically measured experimental values.
20% error bounds in estimates,
Estimated output entropy
Empirically measured output entropy
5
10
15
20

How Do You Choose a Conditioning Function?

Designers can choose their own conditioning function.

- They can screw up…
- …so we assess entropy of conditioned outputs.

90B specifies six “vetted” conditioning functions.

- Cryptographic mechanisms based on well-understood primitives
- Large input and output size, large internal width
- CAN claim full entropy under some circumstances
- No need to run entropy estimates on outputs

The Vetted Functions

- HMAC using any approved hash function.
- CMAC using AES.
- CBC-MAC using AES.
- Any approved hash function.
- Hash_df as described in 90A.
- Block_cipher_df as described in 90A.

- Note: These are all wide (128 or more bits wide), cryptographically strong functions.

Entropy Accounting: Vetted Conditioning Functions

- We don't have to test outputs.
- We know q from table in SP 800-90B.
- We allow the formula to award full-entropy.

- $h_{out} = \text{Output\_Entropy}(n_{in}, n_{out}, q, h_{in})$

Non-Vetted Conditioning Functions

- The designer can choose any conditioning function he likes.
- In this case, we must also test the conditioned outputs.
  - Make sure conditioner isn't too screwy.
- Collect 1,000,000 sequential conditioned outputs.
- Estimate entropy of output.
- **Let h' = the estimate from the conditioned outputs per bit.**

Note: *Designer specifies q in documentation!*

Entropy Accounting: Non-Vetted Conditioning Functions

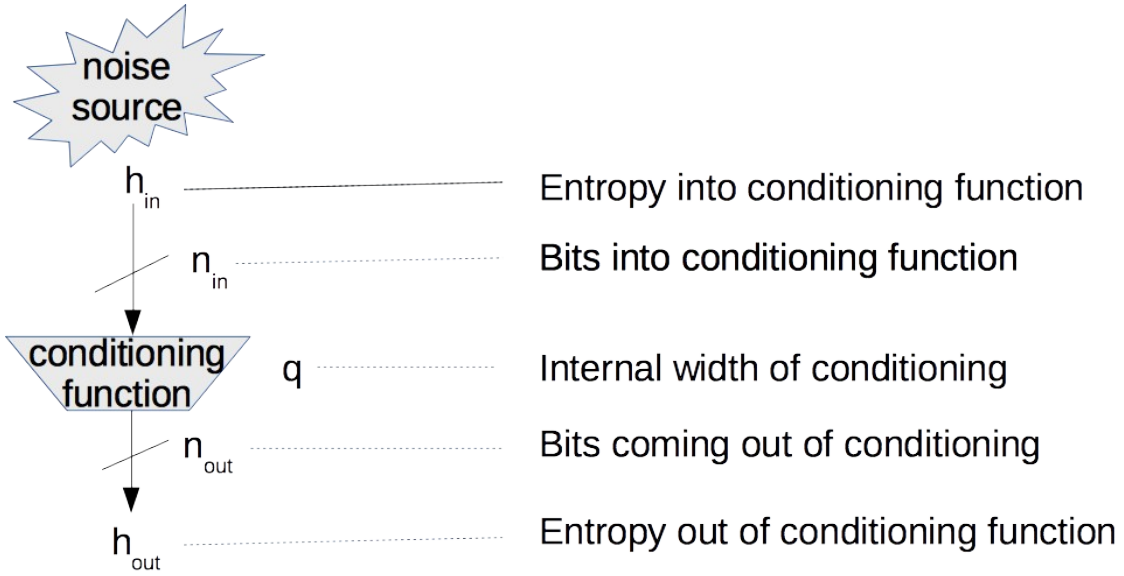
- We compute entropy estimate on outputs:

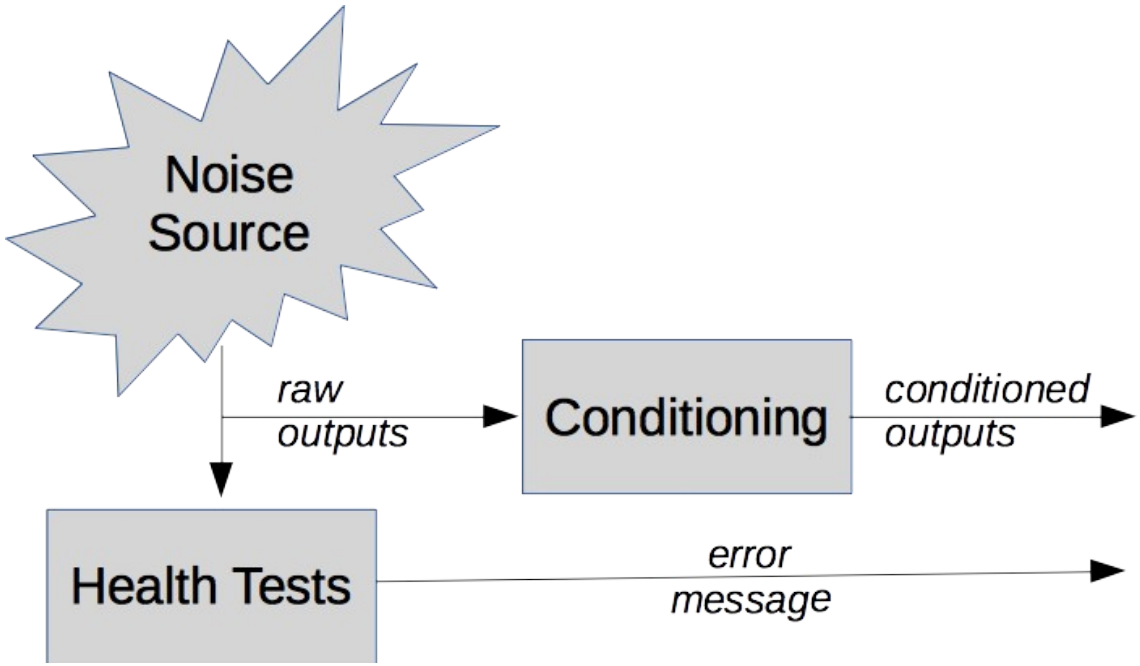
h' = result of entropy estimate (per bit) on conditioned outputs!

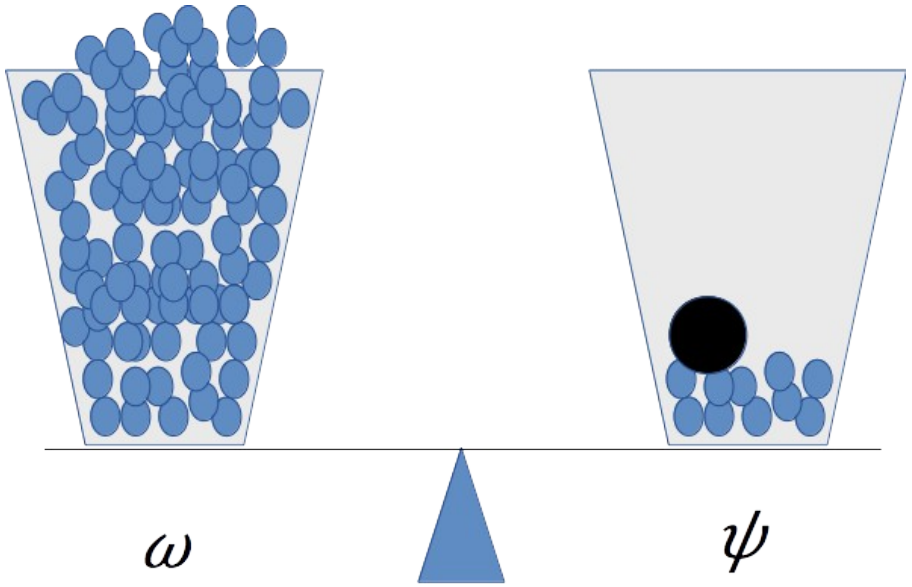
- We don't allow non-vetted functions to claim full-entropy:

$$h_{out} = \min(\text{Output\_Entropy}(n_{in}, n_{out}, q, h_{in}), 0.999\text{n}_{out}, h' \times n_{out})$$

Wrapup


noise source
h_{in}
n_{in}
conditioning function
q
n_{out}
h_{out}
Entropy into conditioning function
Bits into conditioning function
Internal width of conditioning
Bits coming out of conditioning
Entropy out of conditioning function


Noise Source
raw outputs
Conditioning
conditioned outputs
Health Tests
error message


ω
ψ

AIS-31 vs SP 800-90B

- Different meaning of “entropy source”
- We use min-entropy, not Shannon entropy
- 90B entropy source doesn’t have to give high-entropy bits
  - We just want to know how much entropy is in each output.
- We rely far more on statistical testing for entropy estimation
  - Designers still have to give an entropy estimate and justification, but we don’t build the process around a stochastic model.
  - This is largely because of limitations of the FIPS 140 process.
- We require restart tests
- We handle health tests and conditioning are more generically
  - Because we can’t fall back on a stochastic model

90B: Where Do We Go From Here?

- Version 1 of SP 800-90B published Jan 2018
  - We know it still needs work
- Continuous improvement plan
  - List of known issues--working on improvements
  - Short-term: Errata and corrected document
  - Long-term: Notes and research for next revision
- Next year's ICMC (conference on crypto modules)
  - Vancouver, May 14-17 2019
  - One day focused on random bit generation
  - Hoping to gather feedback on 90B from this conference!

Entropy Estimation in 90B

- We over-rely on statistical testing
  - Because we can't expect FIPS 140 validation labs to do analysis well
- **Question:** Can we get something more like AIS31 stochastic models?
  - Without requiring much more expensive and capable validation?
  - Talking about automating most of labs' algorithm testing away!
- One model: IID track
  - Could we do something similar for some non-IID sources?
- Current non-IID evaluation: throwing lots of estimators at source
  - Better as a sanity check than as a final estimate.
  - …how do we do this better?

Questions?

Comments on SP 800-90:

rbg-comments@nist.gov

Contacting me:

firstname.lastname@nist.gov

Extra Slides

Repetition Count

The repetition count test is an updated version of the "stuck test."

- H = entropy estimate for noise source
- P[max] = 2^{-H} is maximum probability for any value.
- alpha = acceptable false positive probability

$$C = 2 + \text{ceil}(-\lg(\text{alpha}) / H)$$

- Count how many times we see a repetition. If count >=C, fail!

- Memory: previous sample, counter

Adaptive Proportion Test

The Adaptive Proportion Test detects when one value becomes too common.

- W = 1024 (binary sources) or 512 (non-binary)
- Based on entropy/sample (H), W, and acceptable false positive rate, choose cutoff value C.
- Every W samples, the test restarts.

- We take the first sample in the window, and count how many times it appears in the whole window.

- If it appears C or more times, then we detect an error condition.

- Memory: counter, window position, value being counted