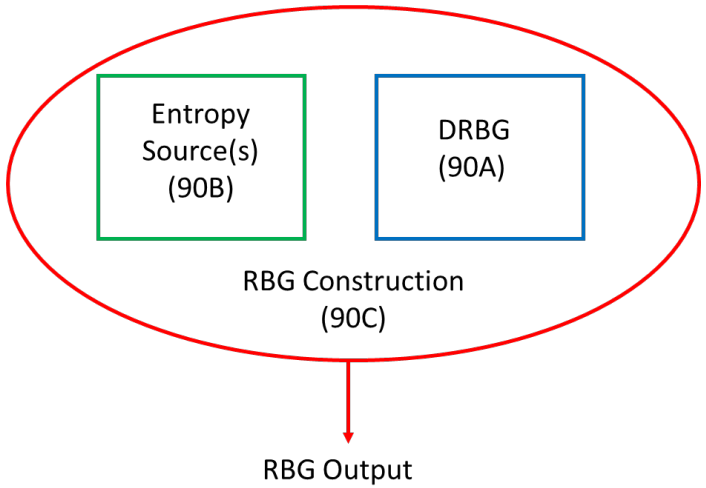
Overview of SP 800–90

John Kelsey
NIST and KU Leuven
November 2024



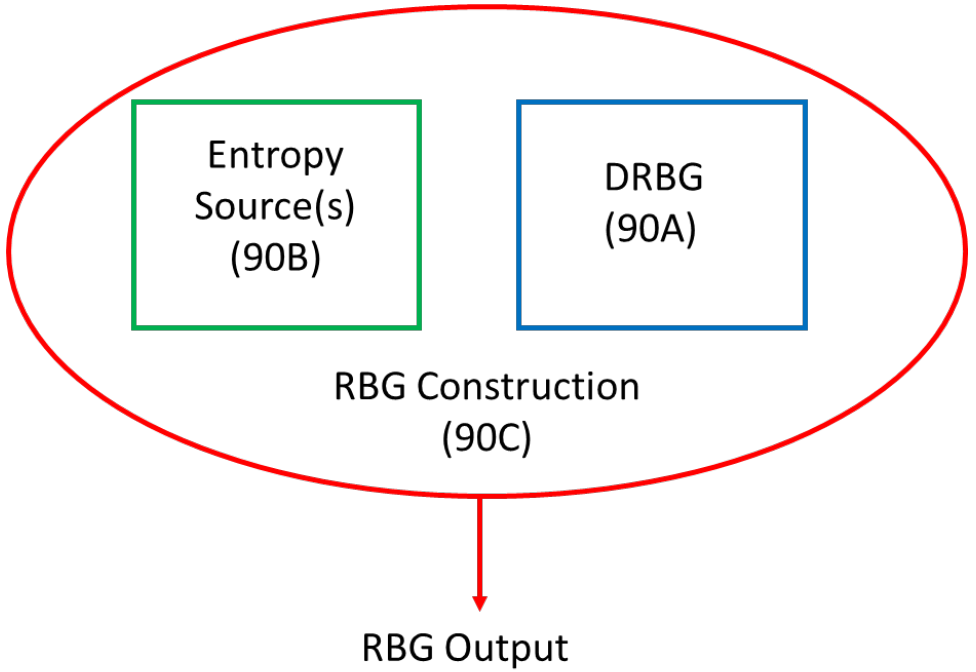
SP 800-90: Big Picture

Entropy source provides seed

DRBG generates cryptographically strong outputs

Rule: Outputs **always** come from DRBG

SP 800-90: A, B, and C



Three parts:

- 90A: DRBGs (Deterministic Random Bit Generators)
- 90B: Entropy Sources
- 90C: Constructing Random Bit Generators

Timetable

SP 900-90A

- First version in 2006
- Multiple revisions since then
- Lots of installed base
- Under revision now

SP 800-90B published 2018

SP 800-90C final draft

- Working through public comments
- Final version out in early 2025

90A: DRBGs


entropy source
seed
DRBG
cryptographically strong pseudorandom bits

SP 800-90A: DRBGs

- DRBG algorithms
  - Deterministic—requires entropy source for seed
- Three standard DRBGs
  - Cannot define your own DRNG
- Security strength (128, 192, 256)
  - AIS 20/31 now requires ≈ 256 bit security
  - We will still allow lower security strengths

DRBGs

CTR-DRBG

- Based on AES block cipher
- Security based on AES key size (128,192,256) + entropy

HMAC-DRBG

- Based on HMAC PRF (SHA2, SHA3)
- Security 128,192,256: based on entropy

Hash-DRBG

- Based on hash fn (SHA2, SHA3)
- Security 128,192,256: based on entropy

A DRBG knows how to do three things:

Instantiate(seed, personalization_string)

- Create new DRBG instance, ready to use

Reseed(seed, additional_input)

- Refresh DRBG state to recover from compromise

Generate(length, additional_input)

- Produce random-looking output

The optional additional inputs

Personalization string and **additional input** are *optional*

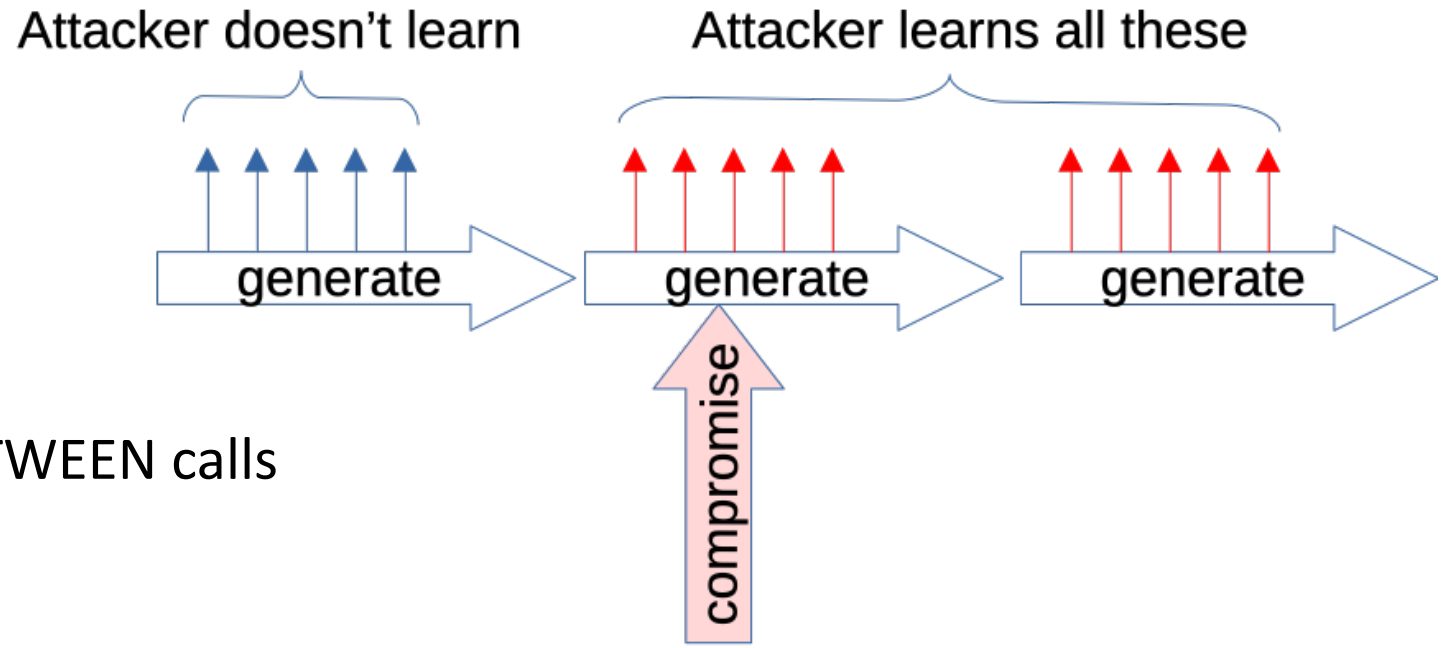
- No assumptions made about contents
- May be adversarially chosen
- Must not weaken security of DRBG
- Should improve security when they contain entropy

A module need not use or even support these optional inputs

Backtracking resistance and Enhanced backward secrecy

DRBGs guarantee enhanced backward secrecy *between* generate calls.

- Expensive “gating” operation between generate calls
- Enhanced backward secrecy (AIS20 definition) provided by all three DRBGs BETWEEN calls



Changes to SP 800-90A

Entropy sources and randomness sources

Current 90A:

- Instantiate and reseed draw from entropy source
  - Fresh entropy required!

- 90C and Next 90A:
- Instantiate and reseed draw from EITHER
  - Entropy source
  - RBG with equal or greater security strength

Instantiation

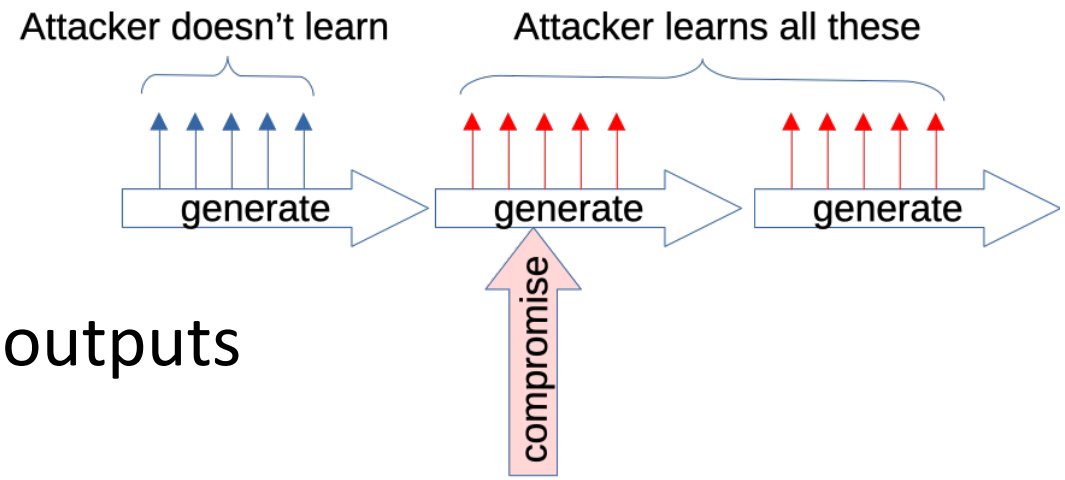
Current 90A: Instantiating DRBG with s bits of security

- Entropy input (s bits entropy) + nonce
  - Nonce MAY be string with s/2 bits entropy

90C and next 90A:

- Randomness input (from entropy source OR RBG)
  - Entropy source: 3s/2 bits min-entropy
  - RBG: 3s/2 bits output, RBG must be same or higher strength
- No more nonce

Problem: long outputs from generate()



- Generate calls currently allow *huge* outputs
  - 2^{19} bits
- Potential for side-channel attacks
  - Especially for AES CTR-DRBG!
- Backtracking resistance only BETWEEN generate calls
  - Compromise state in middle of generate == get all outputs

Planned changes

- Require outputs not leave module until generate completes
  - Avoid keeping a DRBG state in middle of generate
  - This has really been done in modules
  - "Generate function SHOULD complete in a short period of time."

- Considering smaller output limits in next 90A revision
  - Maybe 2048 bits?
  - Can still get many bits, just multiple generate calls

The reseed interval

Current limit 2^{48} generate calls

- Artifact from original analysis of DRBGs
  - Goal: limit outputs to no more than 2^{64} bits or bytes
- Doesn't really accomplish any security goal
- *Going away*

Still considering requiring periodic reseeds in some constructions (90C)

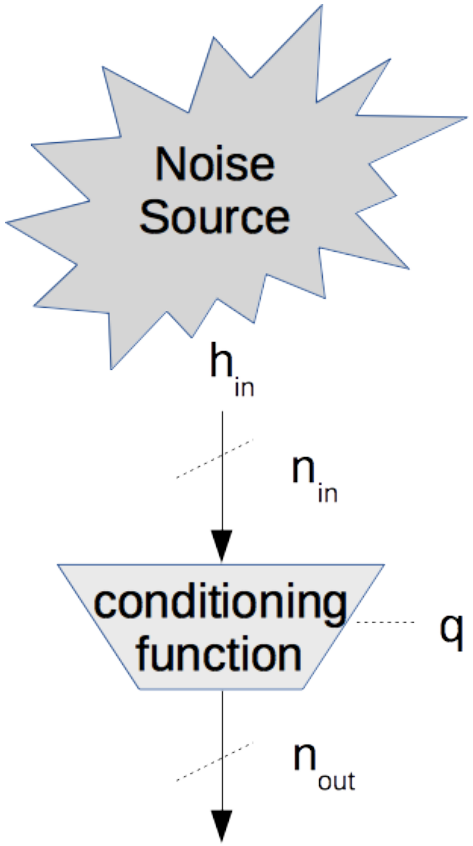
XOF-based DRBG

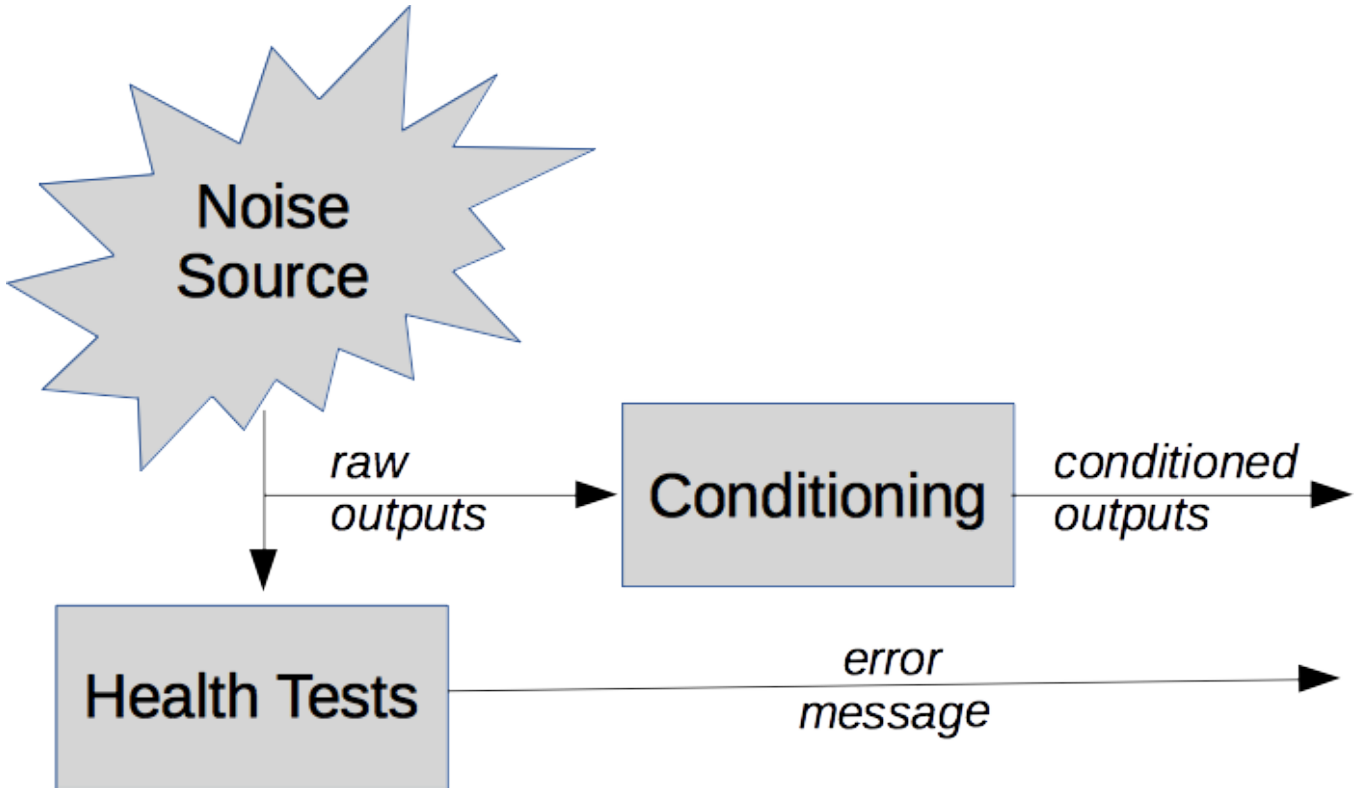
- Planning to add a DRBG based on any XOF
- Example: Shake256, Ascon-XOF

- XDRBG: Design published in ToSC in 2024
- Comes with security proof
- Designed to work with SP 800-90 and AIS 20/31 requirements
- Got feedback and review on paper from BSI*

* Thanks, Johannes!

SP 900-90B: Entropy Sources

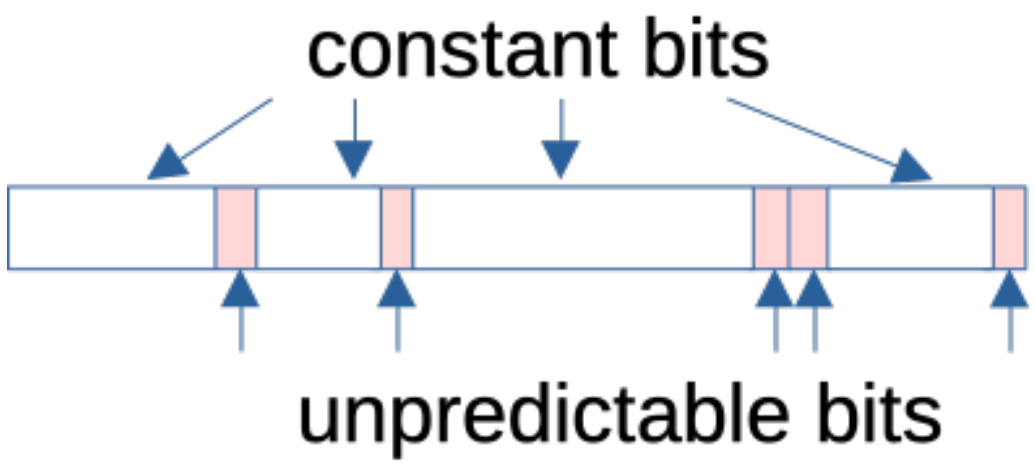


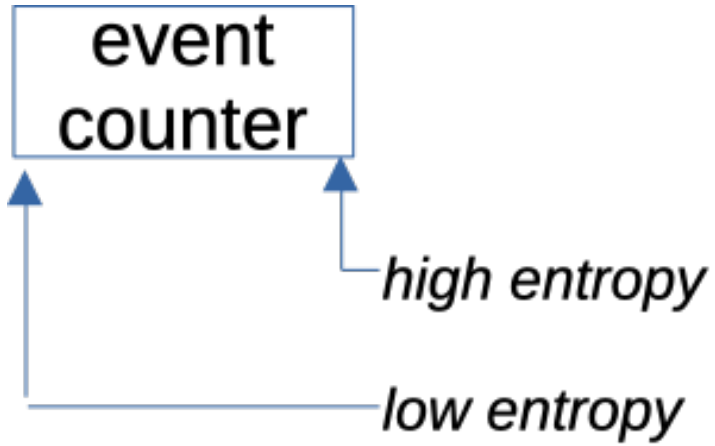


What is an Entropy Source?

SP 800-90B is about how to build an *entropy source.*

- *Produces bitstrings*
- *Tells you how much min-entropy they have*
- *No guarantee on how entropy distributed in output.*





Components of an Entropy Source

Noise Source

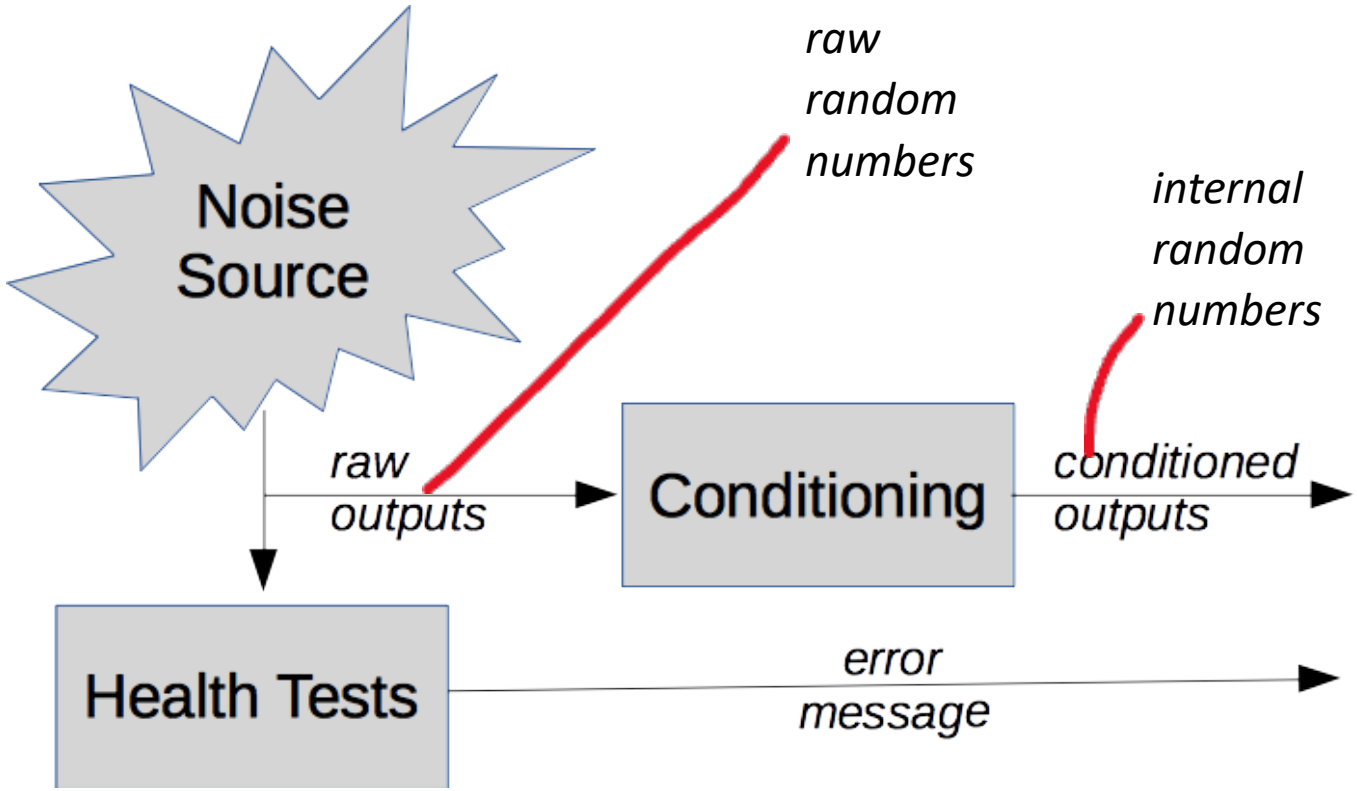
Where the entropy comes from

Health tests

Verify the noise source is still working correctly

Conditioning

Optional processing of noise source outputs before output.

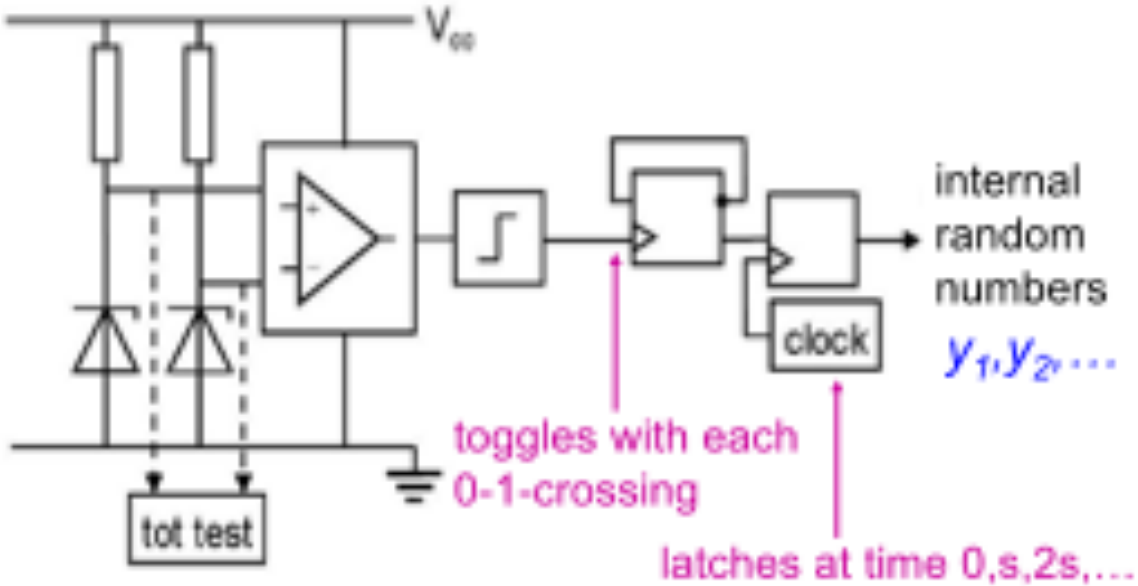


Reminder: An entropy source provides bitstrings with known entropy/sample

Noise Source

- Provides bitstrings with some inherent unpredictability.
  - Don't need high entropy/bit
  - Do need to know how much entropy* we are getting

- Physical OR non-physical
- Examples:
  - ring oscillators
  - interrupt timings
  - noisy diodes + counting



* Min-entropy, not Shannon entropy!

Entropy estimation

- Submitter has to provide entropy estimate and justification
  - Stochastic models not yet required
  - Future: required for physical sources
- Automated tests for entropy estimation*
  - Relied on too heavily now—should be a sanity check
  - Applied to raw bits / raw random numbers
- IID vs non-IID track
  - If designer claims source is IID….

* Next version of 90B will require stochastic models for physical noise sources!

IID track

- Submitter claims IID source
  - Only evaluate as IID if it is claimed
- Submitter provides justification for IID claim
- IID tests try to falsify claim of IID
  - Permutation tests
  - Chi-square test

Deriving a number

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.

+ restart tests → another entropy estimate

Final result is minimum of submitter claim and test results

Health Testing

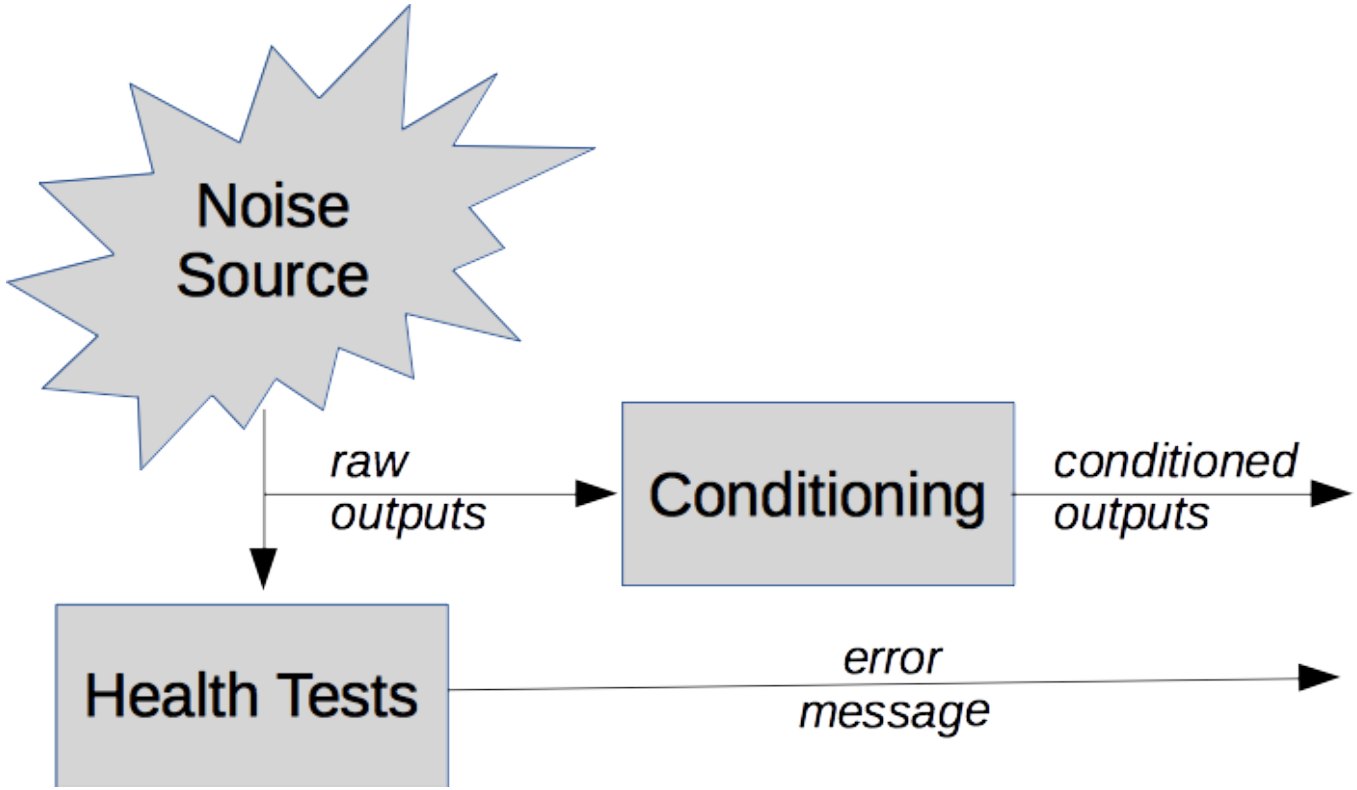
Noise sources are fragile

Failures can be invisible

- Conditioning can hide flaws
- Outputs go into DRBGs
- … nobody ever sees them

Failures → security vulnerabilities

- Improperly seeded DRBG
- Predictable keys



Health tests are essential for security—mandatory in SP 800-90B

Health Tests: Continuous / Startup / On Demand

Continuous Tests

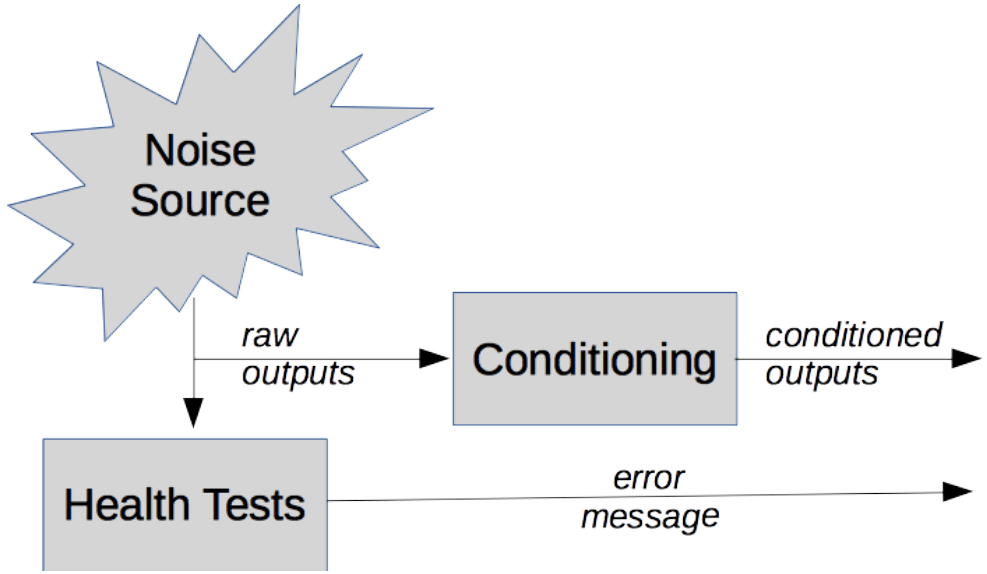
- Going on all the time behind the scenes
- 90B requirements mostly here

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



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

Entropy/sample + false positive rate → cutoff values

Our tests meet minimum requirement

- This will change in next version of 90B
- Designers should understand their sources much better than we can.
- Should design health tests tuned for source
  - 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

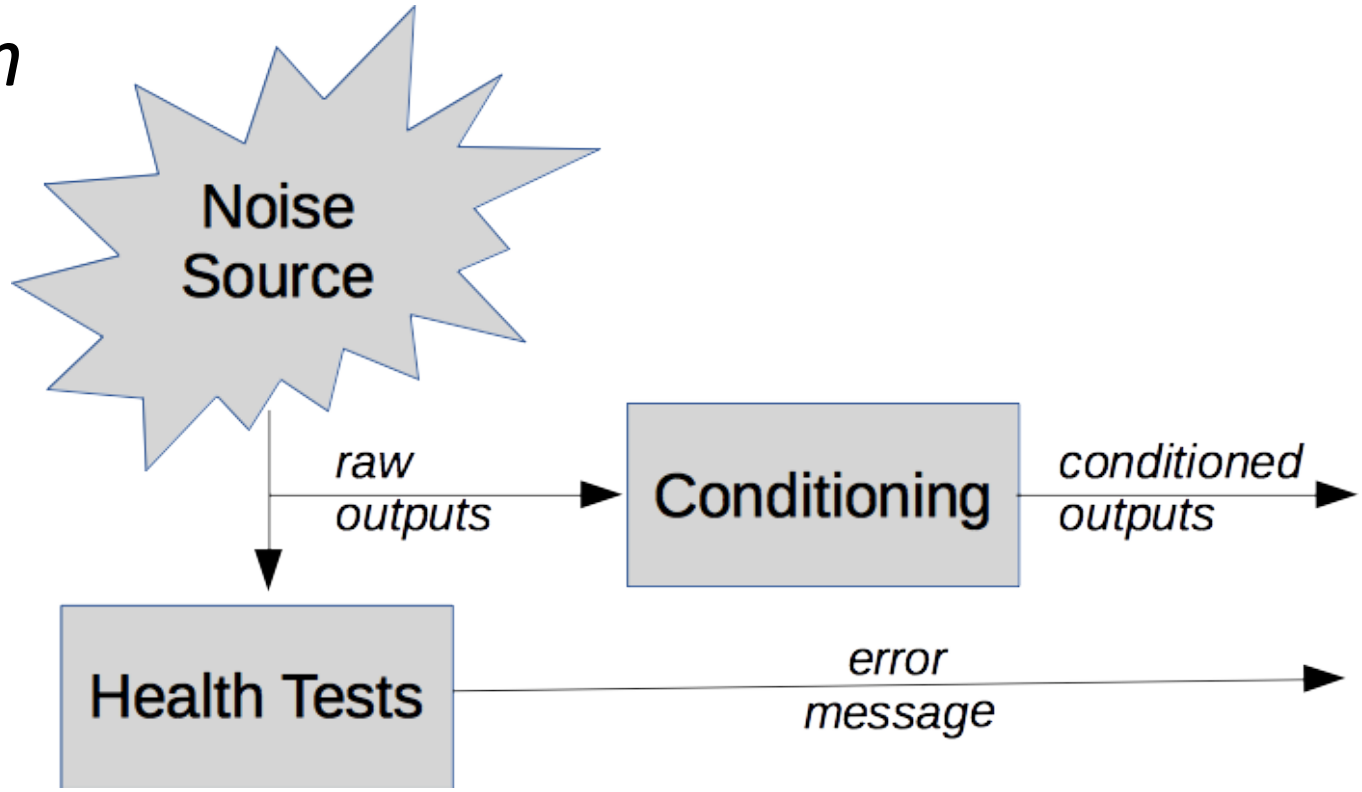
Health Tests: Next version of 90B

- Better requirements for health tests
- No more "safe harbor" by using our tests
  - We never expected these to be what everyone used
- Base tests on parameters of stochastic model + failure modes of source

Conditioning

Post-process raw bits from noise source

- Optional
- Improve statistics
- Compress → improve entropy/bit
- Spread entropy through bit string



90B does not require entropy source outputs to be high entropy

Vetted Conditioning Functions

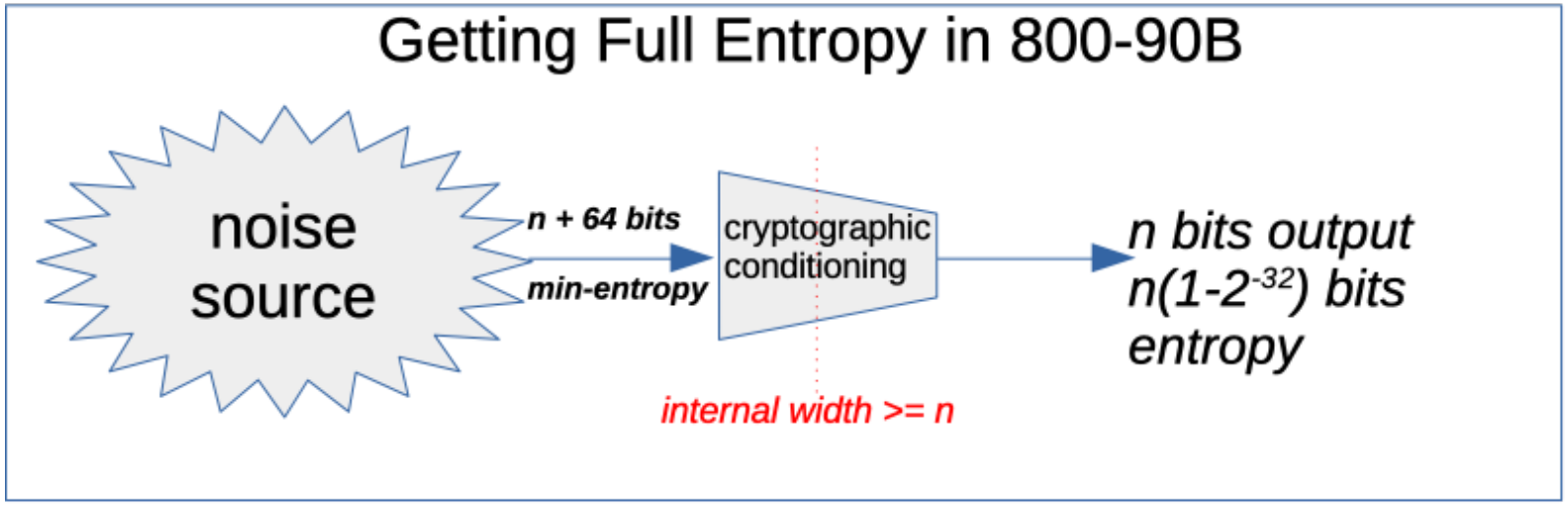
90B specifies six “vetted” conditioning functions.

- 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
- Usually no state between calls

Future plan: Have external list of vetted conditioning functions so additions don't require revising standard!

How to get full entropy in 90B

- Noise source with known entropy/symbol
- Vetted conditioning function
- At least n+64 bits min-entropy in → n bits output



Non-Vetted Conditioning

- Anything not on our list
- Generally non-cryptographic
- Require some justification that these don't interact badly with source
  - Pretty minimal requirement
  - Count on statistical tests to detect problems

Next 90B version:

- Justification based on stochastic model (physical sources)
- Should justify output entropy claim!

Entropy accounting

If you are not getting full entropy….

Vetted conditioning function:

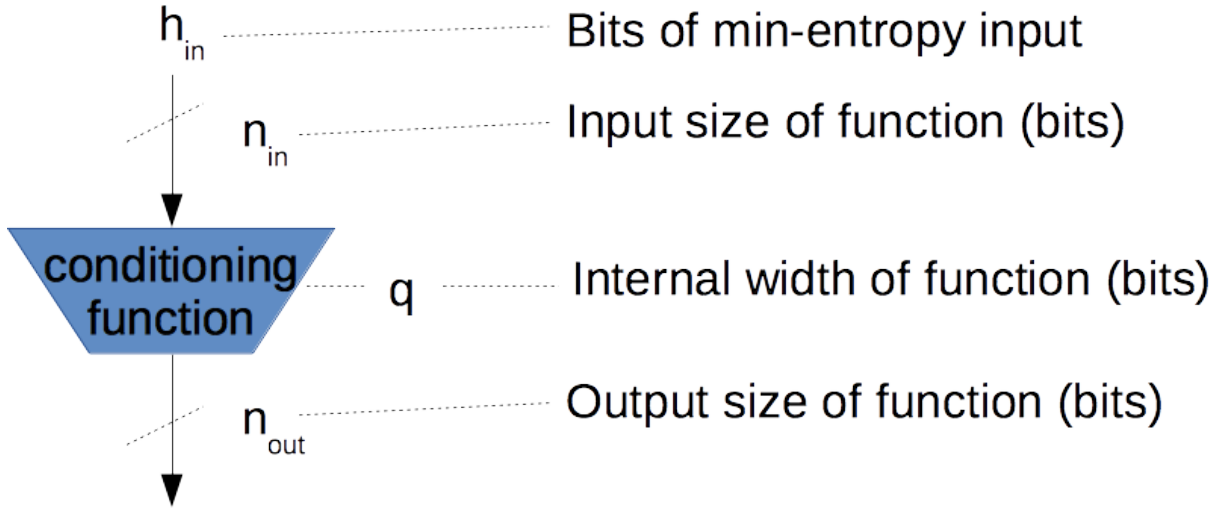
- Use output entropy formula

Non-vetted conditioning function:

- Use output entropy formula
- Run tests on conditioned outputs

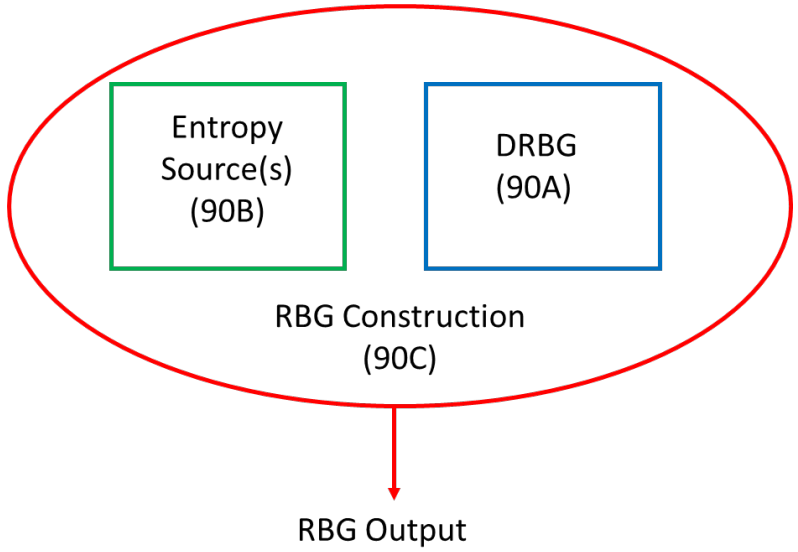
Output_Entropy$(n_{in}, n_{out}, q, h_{in})$:

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



*The formula used to generate Output_Entropy() is adapted from the formulas provided in [RaSt98].

SP 800-90C: Constructions



90C: RBG Constructions

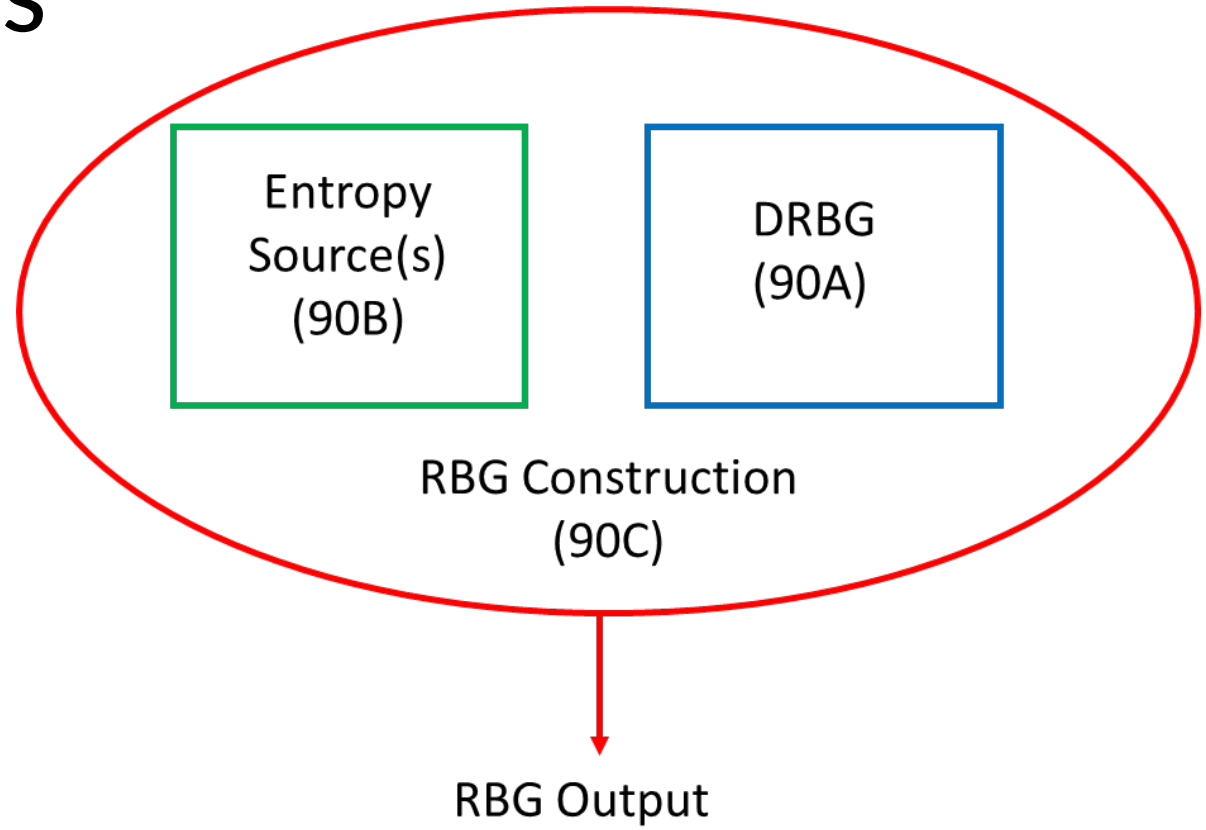
- 90A: DRBG
- 90B: Entropy source
- 90C: Random Bit Generator (RBG)

Specifies four *RBG constructions*

- Different performance/security trade offs

Also specifies other things

- Full entropy requirements
- External conditioning
- Multiple entropy sources



RBG Constructions

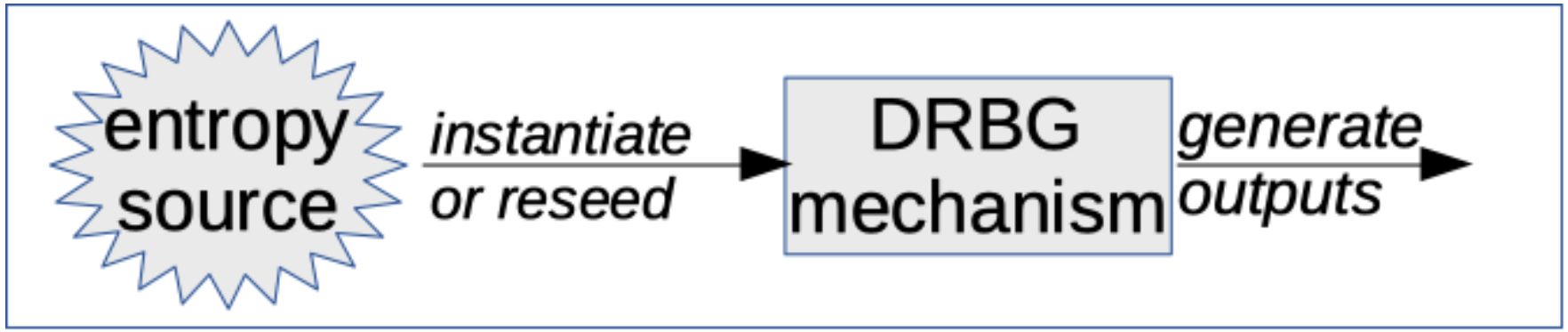
- RBG1 = externally seeded DRBG
- RBG2 = internally seeded DRBG
- RBG3 = full-entropy RBG
- RBGC = chain or tree of DRBGs originating w/ entropy source

Different engineering requirements

Different security and performance traits

Correspond approximately to AIS 20/31 functionality classes

RBG2: Internally Seeded DRBG



- Entropy source + DRBG mechanism
- Internal entropy source
  - Doesn’t need persistent state
  - MAY support reseed on demand
  - MAY reseed automatically “behind the scenes”
- Fixed security strength: {128,192,256}-bits

RBG2(P) and RBG2(NP)

RBG2(P)

- Physical entropy source
- $\approx$ DRG.4

RBG2(NP)

- Non-physical entropy source
- $\approx$ NTG.1 (+ DRG.3?)

RBG3: Full Entropy RBG

- RBG with **full-entropy outputs**

- Supports ALL security levels

  IF entropy source working as expected

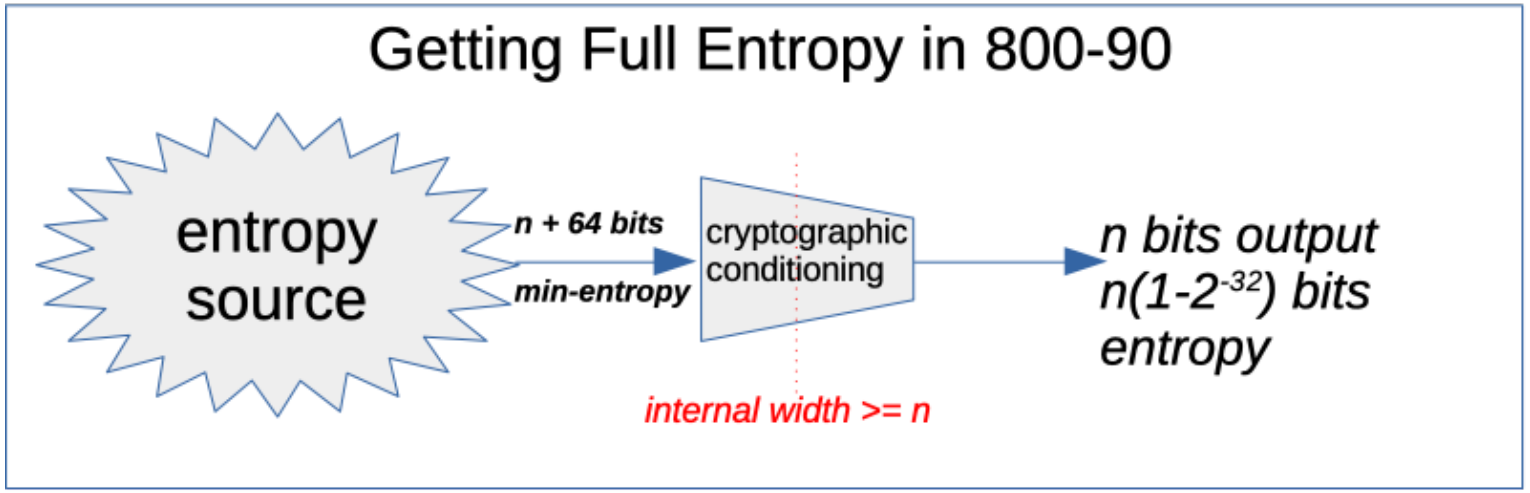
  Full entropy outputs = perfect security

  IF entropy source fails

  Fall back to DRBG Mechanism = computational security

- Always based on physical entropy source

Full Entropy???



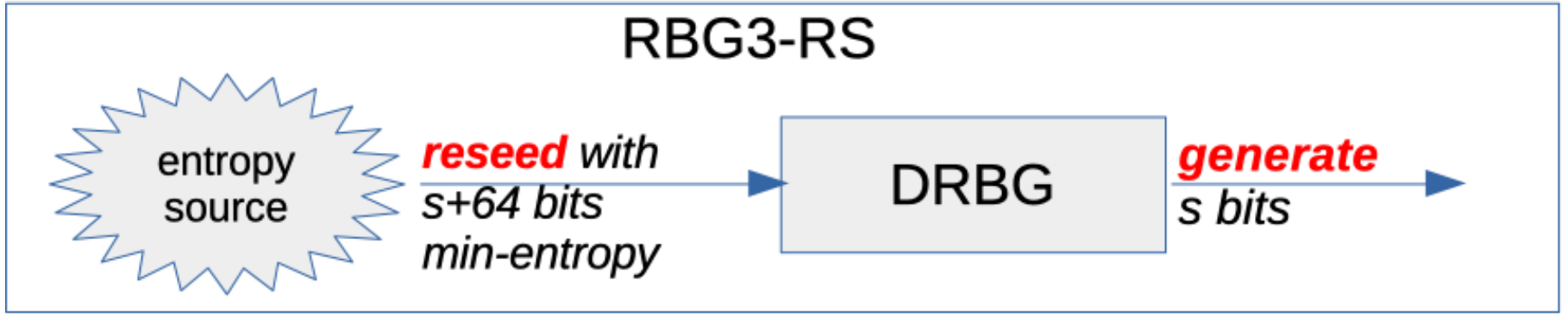
- Each bit of output has $(1-2^{-32})$ bits min-entropy

- Given 2^{64} output bits, can't distinguish from ideal random
  - Even with unlimited computation

Minimal trust of cryptographic primitives

See NIST-IR 8427 for justification and analysis

RBG3-RS



- Start with RBG2 (internally-seeded DRBG)
  - With physical entropy source
- Reseed between each output
  - s+64 bits entropy in, s bits out
- ≈PTG.3

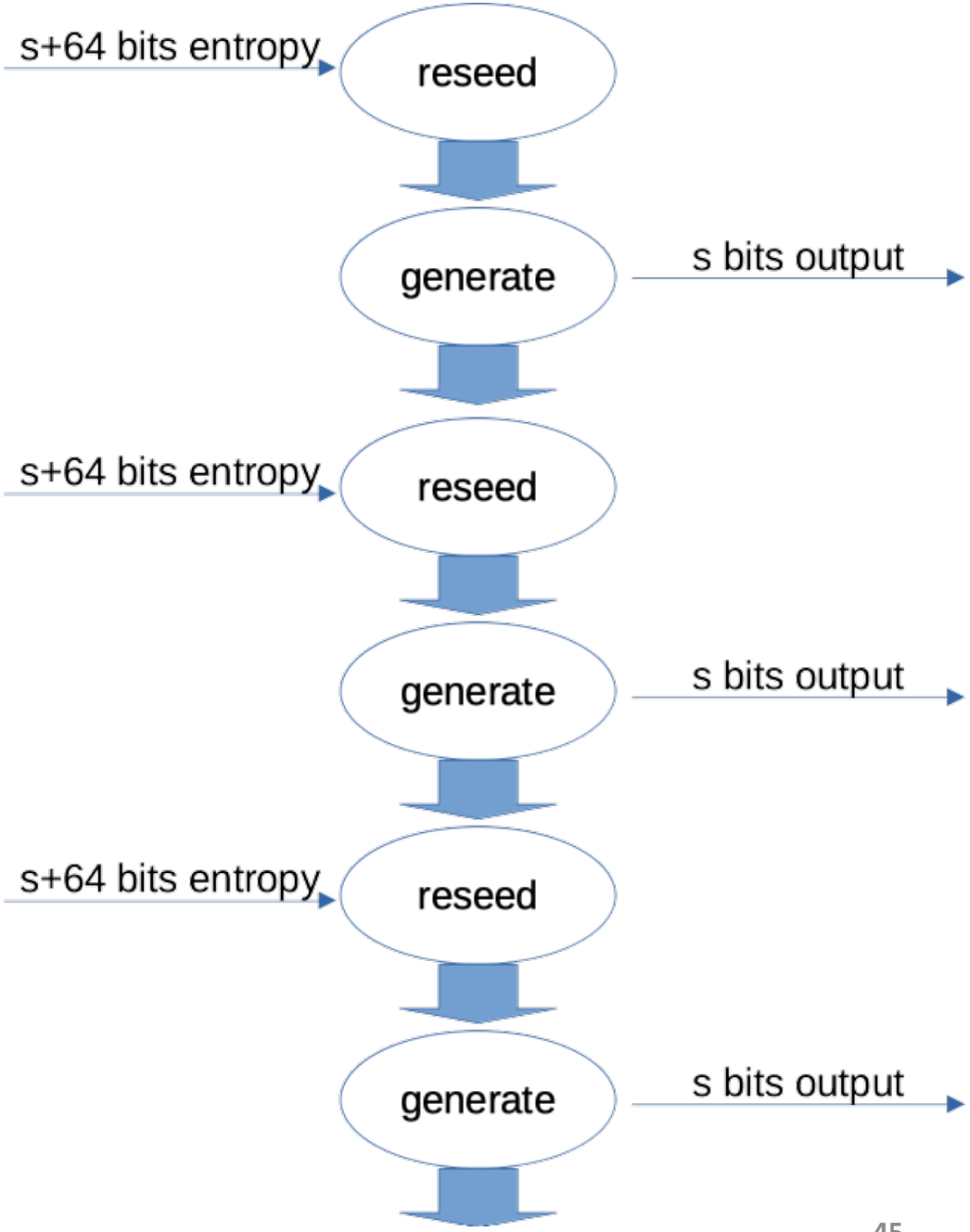
RBG3-RS in detail

DRBG with s bit security strength:

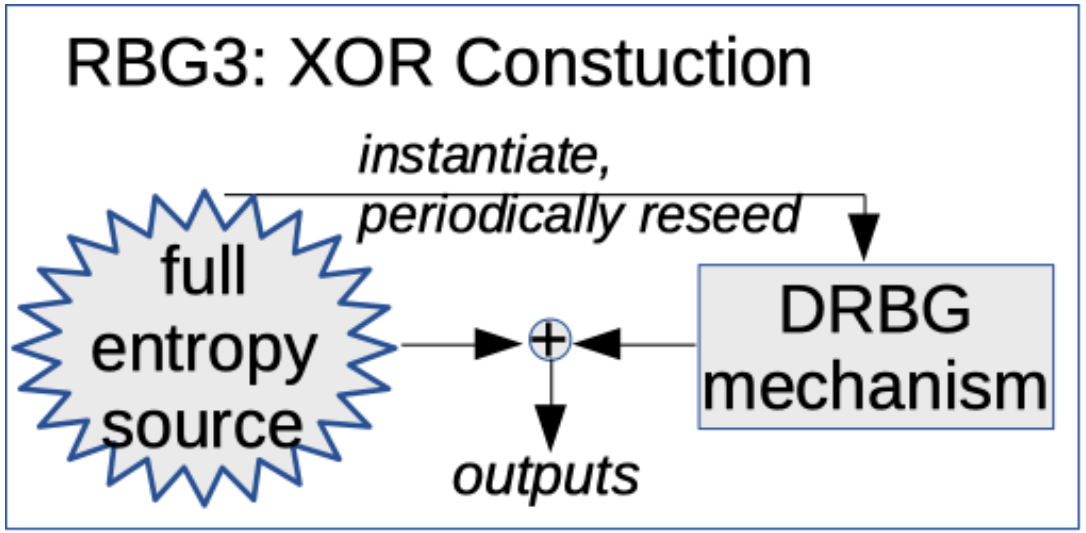
For each s bits required:

- Reseed with s+64 bits entropy
- Generate s bits output

RBG3-RS call generate arbitrary number of output bits in this way!



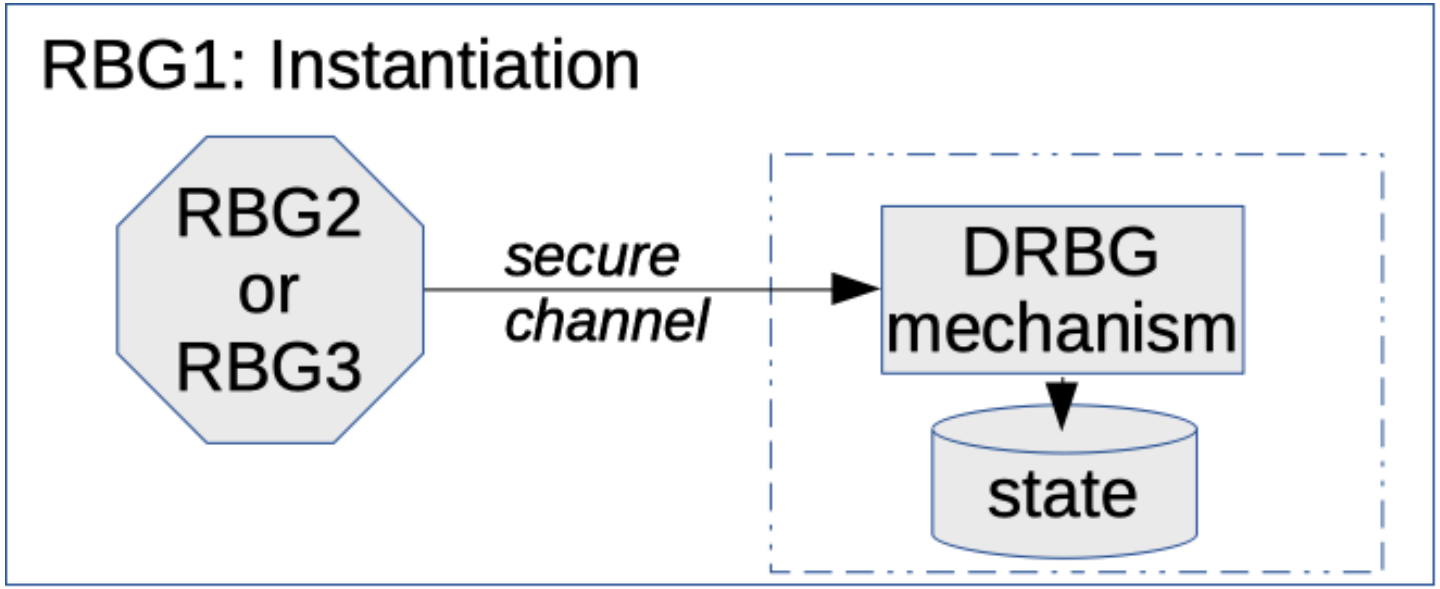
RBG3-XOR

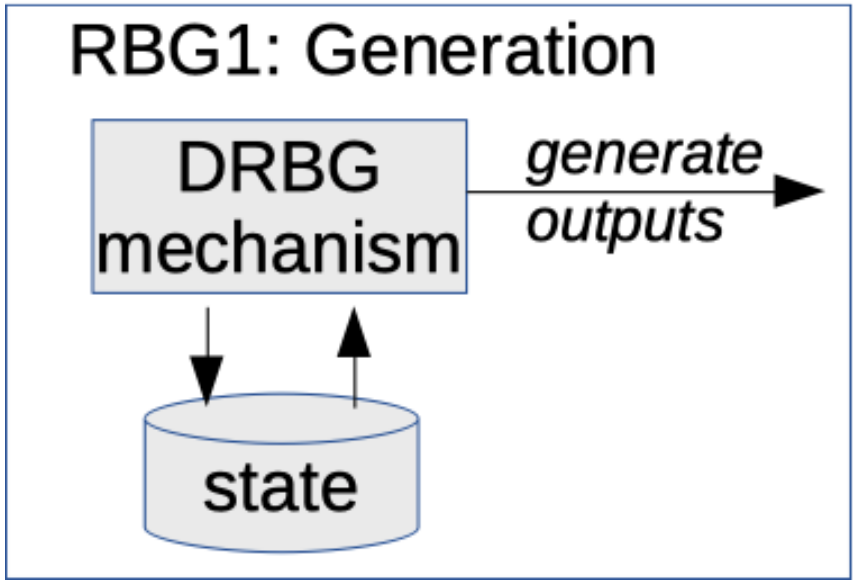


- Start with a full entropy source
  - *Outputs indistinguishable from ideal random bits*
  - *n bits of output have n-2*$^{-32}$ bits of min-entropy
- Seed DRBG mechanism
- Outputs = DRBG output XOR Full Entropy Source output
- Can also get normal DRBG outputs
  - Just generate from DRBG mechanism

≈ DRG.4 (physical entropy source + DRBG)

RBG1: Externally Seeded DRBG





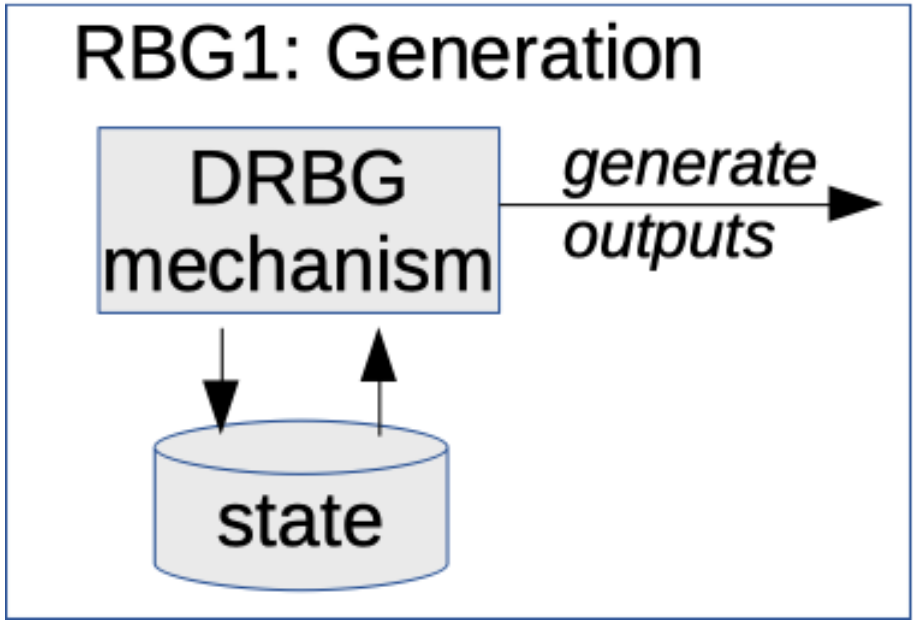
- Instantiated once **from outside module**
  - MUST be seeded from RBG2(P) or RBG3
  - **Physically secure** channel needed
  - RBG2 at least as high security strength as RBG1

Note: These are all documentation requirements—can't be tested!

RBG1 (cont'd)

Instantiates once

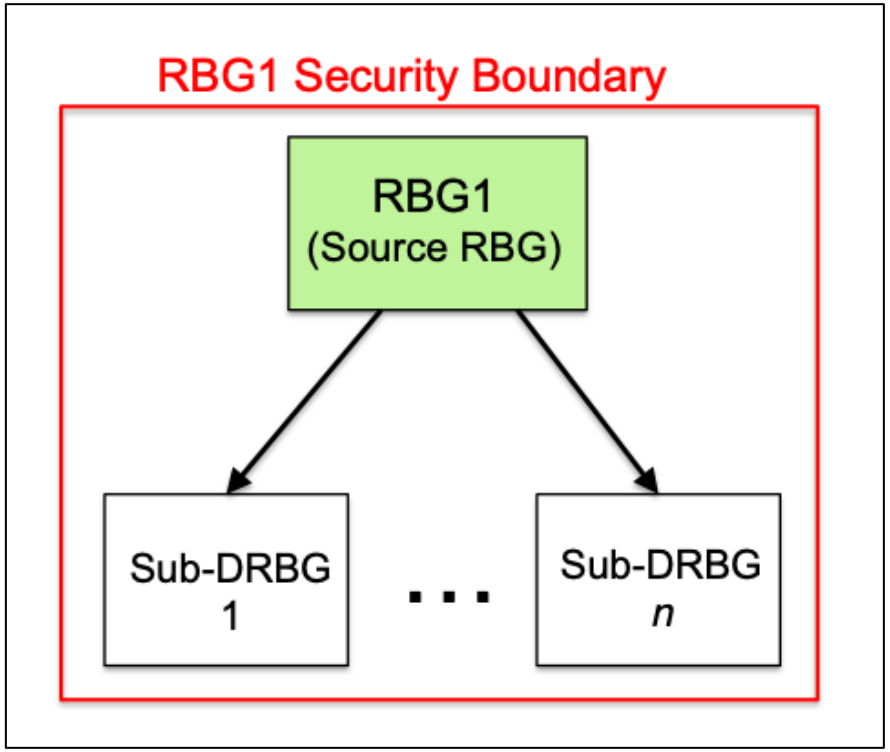
Never reseeds



- No onboard entropy source
- Requires persistent memory
  - …to keep constantly-evolving DRBG state
- Memory must not reveal previous states!
- Cannot recover from a compromise
- Fixed security strength: 128, 192, 256.

≈ DRG.3

RBG1 Sub-DRBGs

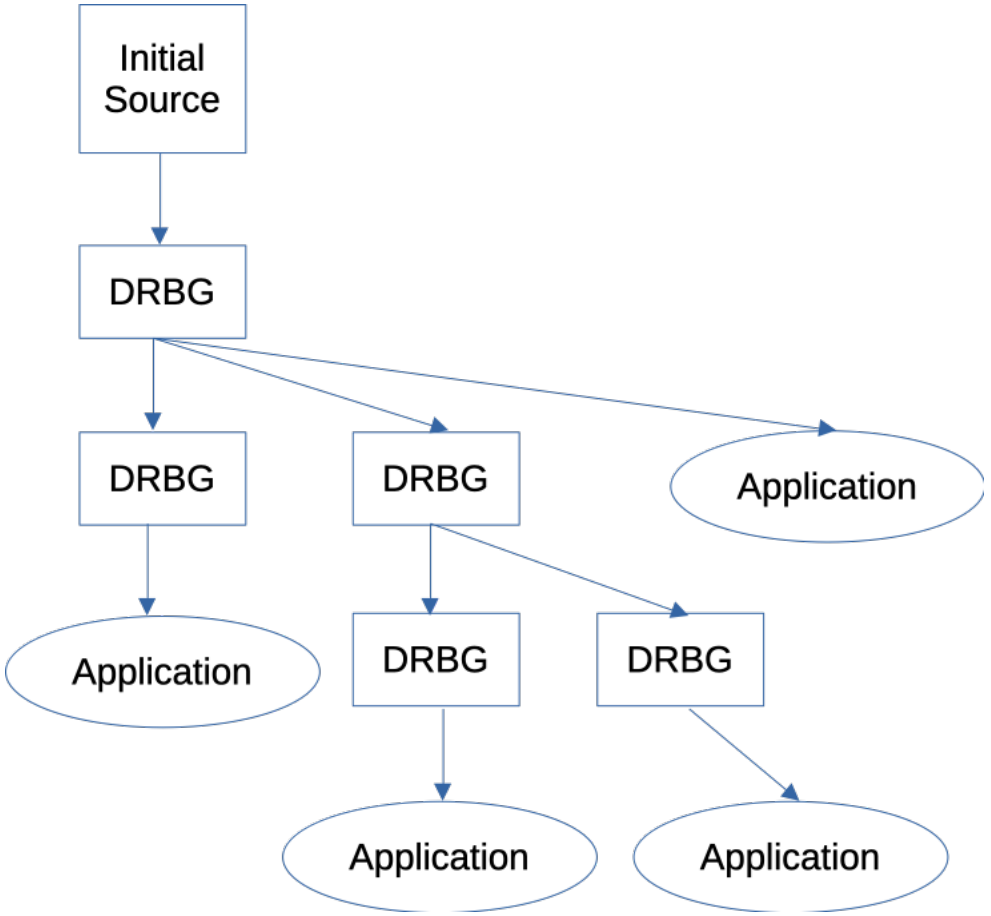


- RBG1 construction can instantiate one layer of subordinate DRBGs (Sub-DRBG)
- Sub-DRBGs reside in the same security boundary as the RBG1 source
  - Use the same DRBG mechanism
- Sub-DRBG output shall not provide input for the RBG1 source

RBGC

- Each DRBG:
  - Has exactly one seed source
  - May seed multiple DRBGs
  - May provide bits to applications
  - Cannot seed a DRBG with higher security than its own
- Initial source: Ultimate root of trust
  - RBG2(P)
  - RBG2(NP)
  - RBG3
  - Full entropy source

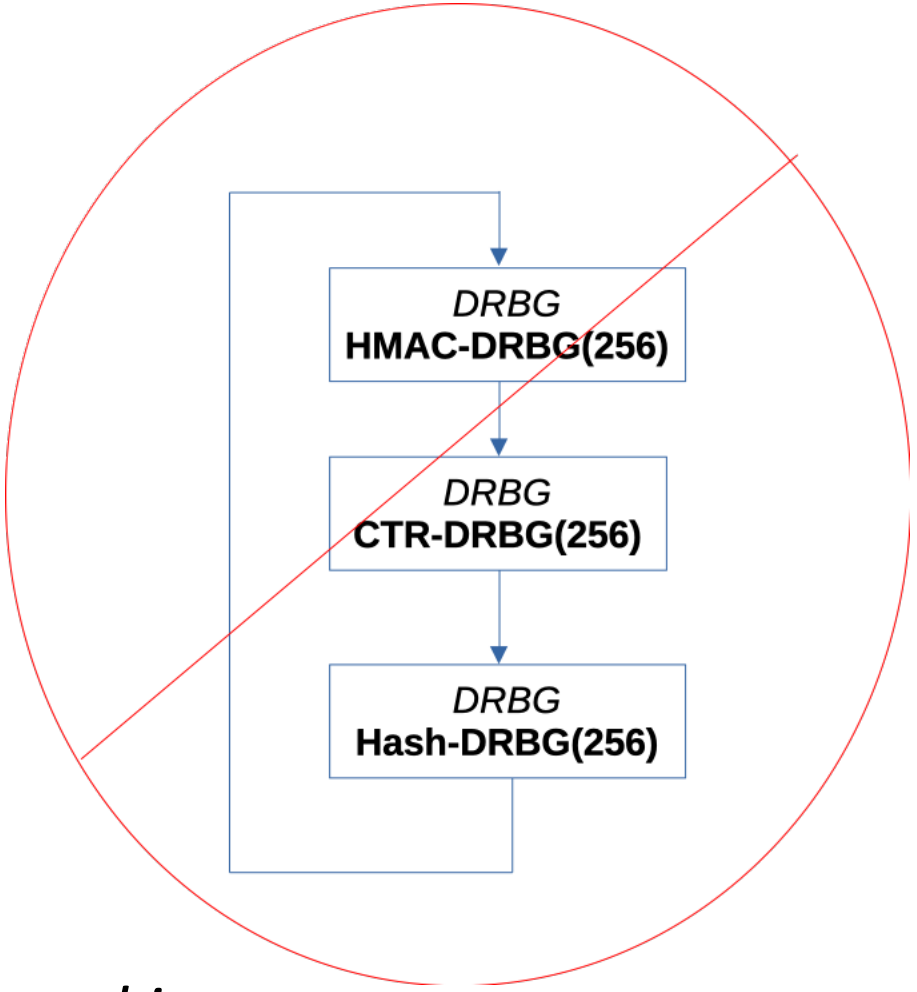
≈ DRT.1



Loops are not allowed

Example: RBGC with loop

- No entropy, no security
- No DRBG may be part of its own seed source!



This leads to somewhat complicated rules and issues

NIST/BSI Collaboration

NIST/BSI Collaboration

Started with SP 800-90B in 2016-2018

- We were standing up entropy source evaluation
- They'd been doing it for >10 years
- They gave us lots of feedback
- Even came to NIST to help us finalize requirements

We got a *huge* benefit from their experience, even when we couldn't copy everything they were doing

Ongoing collaboration

Work on aligning requirements where possible

- Not always easy: our processes are very different!
- Especially focused on incompatibilities...
- ...e.g., we require you do X, they require you do not-X.

Online meetings (currently every 2 weeks)

- In-person meetings, joint presentations, joint publications

SP 800-90C and AIS 20/31

- NIST working on finalizing SP 800-90C
- BSI working on updating AIS 20/31
- Opportunity to avoid conflicts in standards ”from the beginning”
  - MUCH easier to fix before standard is finalized
  - Reflected in 90C and new AIS 20/31, especially RBGC / DRT.1

Note: 90C RBG constructions ≈ AIS 20/31 functionality classes

New Joint Publication: NIST IR 8446

Bridging the Gap between Standards on Random Number Generation

- Comparison of SP 800-90 Series and AIS 20/31
- Discussion of RBG constructions/functionality classes
- Focus on how to comply with requirements of NIST and BSI at once
- **Goal**: help vendors comply with both standards in same design!

https://csrc.nist.gov/pubs/ir/8446/ipd

Open for public comment until Dec 20, 2024

Wrapup

- SP 800-90
  - 90A: DRBGs (DRNGs)
  - 90B: Entropy Sources (PTGs and NTGs)
  - 90C: Constructions (Functionality classes)

- Working with BSI to harmonize with AIS 20/31
  - Goal: Make it workable to comply with both
  - Standards will never be identical...
  - ...but we can avoid contradictions between standards

Questions?