# Random Bit Generation

## Theory and Practice

**Joshua E. Hill**

Department of Mathematics, University of California, Irvine

Math 235B
January 11, 2013
`http://bit.ly/XWdBtv`
v1.14

UNIVERSITY *of* CALIFORNIA · IRVINE

# Talk Outline

## Section 1

# Introduction

# The Story Thus Far

- We have seen many, many uses for random numbers in cryptography.
- This is for reasons coming from game theory.
  - By Kerckhoffs’ principle, we assume that adversaries know the design, thus know how secret values are selected.
  - Game theory tells us that in these circumstances, the random selection of parameters yields the least advantage for the attacker.
- Within computers, we represent any number as a sequence of bits, so I’ll generally use the term *random bit generator*.

# Random is as Random Does

- *Random* is not a characteristic of a number.
- *Processes* are random, and we refer to the numbers produced by such processes as random numbers.
- Such numbers can be modeled as random variables selected from some probability distribution.

# Wheels within Wheels…

The term *random bit generator* is used in a few distinct ways:

1. Truly random: Derived from some underlying physical phenomena which is unpredictable absent direct measurement.
2. Cryptographically random: Computationally difficult for an attacker to guess future outputs given past outputs.
3. Statistically random: Models some particular statistical distribution well.

# When *I* use a word…

## Definition

A cryptographic random bit generator, with security bound $L$ bits, produces sequences of random bits $(R_1, R_2, \ldots, R_n)$ such that

1. The generator is unbiased: $\Pr\left(R_j = 0\right) = \frac{1}{2}$.
2. The bits are uncorrelated: $\Pr\left(R_j = 0 | R_1, R_2, \ldots, R_{j-1}\right) = \frac{1}{2}$.
3. Negligible advantage: An attacker can't distinguish between a true uniform random bit generator and the cryptographic random bit generator without performing at least $2^L$ operations.

This third goal is equivalent to the goal "Computationally difficult for an attacker to guess future outputs given past outputs.".

# You Can't Get There From Here…

There are a few approaches to this:

- Use a non-deterministic random bit generator (NDRBG, a.k.a., a *True Random Number Generator*).
    - Most physical sources aren't well modeled by uniform distributions.
    - Most physical sources are fragile and can fail, often subtly.
    - Most physical sources produce random bits very slowly.
    - Many physical sources can be affected by a suitably powerful attacker.
- Use a deterministic random bit generator (DRBG, a.k.a. pseudo-random number generator, or PRNG). Good designs:
    - Have excellent statistical properties.
    - Are easy to test.
    - Can produce vast amounts of output quickly.
    - Are difficult for an attacker to influence.
    - Can accumulate entropy (uncertainty).

# You Can't Get There From Here...

There are a few approaches to this:

- Use a non-deterministic random bit generator (NDRBG, a.k.a., a *True Random Number Generator*).
  - Most physical sources aren't well modeled by uniform distributions.
  - Most physical sources are fragile and can fail, often subtly.
  - Most physical sources produce random bits very slowly.
  - Many physical sources can be affected by a suitably powerful attacker.
- Use a deterministic random bit generator (DRBG, a.k.a. pseudo-random number generator, or PRNG). Good designs:
  - Have excellent statistical properties.
  - Are easy to test.
  - Can produce vast amounts of output quickly.
  - Are difficult for an attacker to influence.
  - Can accumulate entropy (uncertainty).
  - Require input that cannot be predicted by an attacker. :-(

# You got your NDRBG in my DRBG!

- Reasonable designs must involve both a DRBG and a NDRBG.
  - The NDRBG could be integrated into the design.
  - The NDRBG could be used during manufacturing.
- The NDRBG is the ultimate source of uncertainty (and thus security).
- The DRBG:
  - Conditions the output and gives it excellent statistical properties.
  - Remains secure any time after being seeded by reasonable NDRBG input, even if the NDRBG fails later.
  - Can produce a very large amount of input given a very modest amount of reasonable input from the NDRBG.

## Section 2

# Non-Deterministic Random Bit Generation

# NDRBG Outline

1 Introduction

2 Non-Deterministic Random Bit Generation
- Information Theory
- Entropy Source
- Test, Test, Test
- NDRBG Conclusion

3 Deterministic Random Bit Generation

4 Conclusion

### Subsection 1

## Information Theory

# How Many Bits Would a Bit Compressor Compress if...

- The traditional measure of *uncertainty* from Information Theory is called entropy.
- Much like randomness, *messages* do not have entropy. *Message sources* have entropy.
- There are several related notions of entropy.
- Shannon entropy is the most widely adopted notion of entropy.
- It tells you the minimal average message length for a source.

## Definition

Shannon Entropy

$$H(X) = -\sum_{i=1}^{n} p_i \lg(p_i)$$

# No, Mr. Bond, I Expect You to Guess.

- Shannon entropy is not really what we want.
- We want a *worst case*, not the *average case*.
- We get it from a generalization of Shannon Entropy called Rényi entropy.

## Definition

Rényi Entropy

$$H_\alpha(X) = \frac{1}{1-\alpha} \lg \left( \sum_{i=1}^{n} p_i^\alpha \right)$$

- Letting $\alpha \to 1$ gives us Shannon Entropy.

# Mirror Mirror on the Wall...

- Take the limit as $\alpha \to \infty$ yields the worst-case: *Min-entropy*.
- In some sense, min-entropy is a lower bound for any other notion of entropy.

**Definition**

Min-Entropy

$$H_\infty(X) = -\lg(\max_i p_i)$$

## Subsection 2

# Entropy Source

# When The Diode Breaks: Sources of Entropy

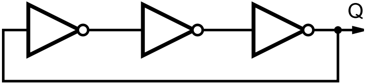
- Well understood sources
  - Ring oscillators
  - Noisy diodes
  - Radioactive decay
  - (Other) Quantum effects
- Somewhat understood sources
  - Fluid turbulence (or other other chaotic systems)
  - Audio noise
  - Radio noise
  - CCD noise
- Poorly understood sources
  - Process scheduling patterns
  - Network packet arrival timing
  - Booting randomness
  - Keyboard / mouse movement

# Beyond Good and Evil

- A Good Source
  - is very simple and easy to analyze.
  - has a readily identifiable and quantifiable source for uncertainty.
  - is difficult for an attacker to monitor.
  - can be well modeled by some well understood statistical distribution so that min-entropy can be estimated.
  - can be easily tested for deviation from this expected distribution.
  - is stable across the the expected operational range of the system.
- In summary, a good source is both *secure* and has *assurance* of security.

# Wait, Am I in the Right Room?

Let's examine an ring oscillator:



Source: Inductiveload via: Wikipedia

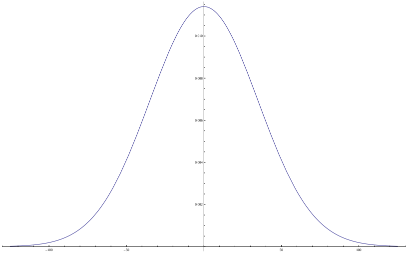
- Each gate has a finite switching time.
- Variation in switching time is called *jitter*.
- This jitter is induced by thermal noise, which is thought to be a random process (quantum effects dominate).
- The jitter for one gate is roughly normally distributed.
- Chaining together multiple gates sums the jitter for each gate.
- The sum of independent identically distributed normal distributions is normal (with the same mean, and larger standard deviation).

# Rings and Things You Sing About, Bring ’em Out

- Initialize the system by opening the loop and allowing to stabilize.
- Induce a pulse whose length is (much) less than the oscillator period.
- Close the loop.
- Time period between rising edge of the pulses.
- Subtract the average oscillator period: this is the jitter.

# Not A Reference to the Book!

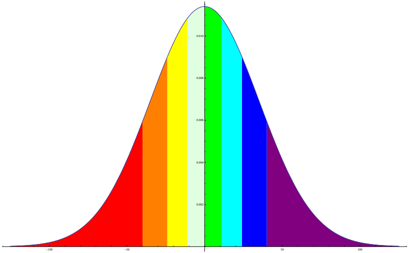
- The probability distribution function (PDF) for one implementation's jitter looks like this:



- Divide the PDF into roughly eighths.
- $p_{\max} = 0.130205$ so $H_\infty(X) \approx 2.94$. Great!

# Not A Reference to the Book!

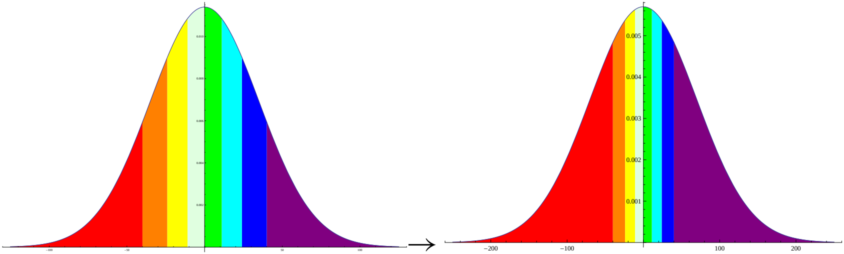
- The probability distribution function (PDF) for one implementation's jitter looks like this:



- Divide the PDF into roughly eighths.
- $p_{\max} = 0.130205$ so $H_\infty(X) \approx 2.94$. Great!

# Hot Space

- What if the chip gets a bit warm?



- $p_{\max} = 0.283855$ is now so $H_\infty(X) \approx 1.82$.
- We now have only 61% of our prior entropy. :-(

# Fix Up

- The assumption about the parameters of the distribution is fragile.
- To make our analysis more conservative, analyze the timing difference between consecutive pulses.
  - If the first is longer, output a $1$. If the second is longer, output a $0$. If equal, no output.
  - Difference of two i.i.d. normal distributions is a normal distribution.
  - Mean and standard deviation should be stable, so $p_{\max} \approx .5$, so $H_\infty(X) \approx 1$ bit.
- The *local* conditions between two consecutive pulses should be very stable.
- Ideally, provide the full timing values as the seed.
- Account only for one bit of min-entropy per pair.

## Subsection 3

# Test, Test, Test

# Design Testing

- After implementation, test your implementation against your assumptions.
- Many tool chains silently remove uncertainty.
- We can produce a set of likely upper entropy bounds given a great deal of seed data.
- Raw timing values allow for extensive design tests.
- If we expect full entropy, testing is “easy”!
  - Diehard / Dieharder
  - sts
  - Statistical tests require some expertise to run and interpret.
- If we expect our data to have less than full entropy, we can only run a subset of these tests, and interpretation must be done very carefully.
- We can estimate (Shannon) entropy with compression tests.
- Testing seed data to assess entropy *must* be conducted prior to any cryptographic processing.

# Unit Testing

- Some statistical testing should continue while in use. Examples:
  - Continuous output testing for a stuck-value.
  - Periodic $\chi^2$ test.
- Tune the probability of false failure to an acceptable level (set $\beta$ low enough so that the lifetime probability of false test failure is low).
- Technically, this reduces entropy, though if $\beta$ is low enough, this is negligible.

## Subsection 4

# NDRBG Conclusion

# Lessons

- Only uses sources that you *really* understand.
    - What physical process is responsible for entropy?
    - What probability distribution models this process well?
    - How does this process change with conditions?
- Try to be very conservative with entropy analysis.
    - This results in high assurance lower bound estimates.
    - Never throw away possible entropy, just account and combine conservatively.
- Test!
    - Verify that your analysis is supported by reality.
    - Verify that the running NDRBG hasn't failed.

## Section 3

# Deterministic Random Bit Generation

# DRBG Outline

1. Introduction
2. Non-Deterministic Random Bit Generation
3. Deterministic Random Bit Generation
    - DRBG Introduction
    - OFB Based DRBG
    - ANSI X9.31-1998 A.2.4 DRBG
    - CTR-DRBG
4. Conclusion

## Subsection 1

# DRBG Introduction

# A Reminder

> *Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number – there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.*
>
> — John von Neumann

# General Idea

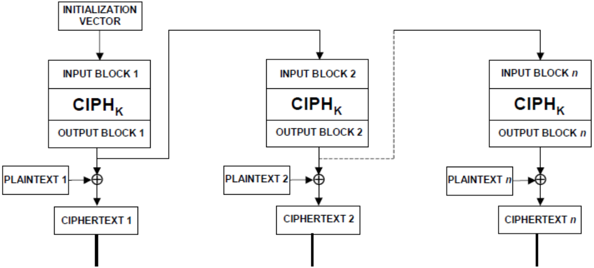
- Conceptually, a DRBG involves a few processes
  - A function that seeds the DRBG.
  - A function that processes the internal state between outputs.
  - A function that outputs random bits ("Generate").
- Seeding requires entropy input. The other functions can optionally accept entropy input.
- Internal state collision leads to cycles (there may be a birthday paradox problem, depending on the design).
- We make use of some cryptographic primitive within each of these functions.
- Any entropy input must be in large blocks (min-entropy at least as large as the security bound).
- Seed input may allow the attacker to manipulate the internal state.

## Subsection 2

# OFB Based DRBG

# A Bad Idea

- DES in OFB mode.



Source: NIST SP800-38A

# A Bad Idea: Notes

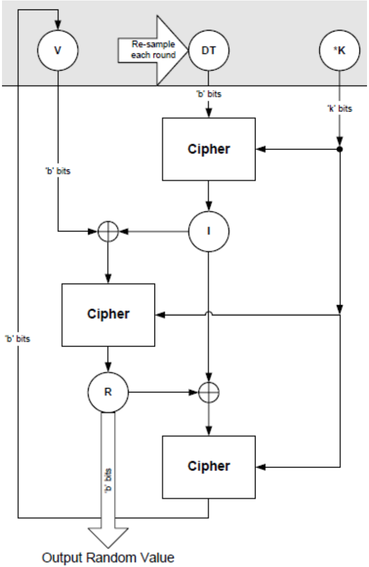
- Seed by selecting the key, $K$, and the one block $IV$.
- Keep $K$ secret, use the output of the DES function as the DRBG output.
- This (mostly) has excellent statistical properties.
- Problem: We expose our internal state (as the DRBG output).
- Problem: only $V$ is updated. $K$ is fixed.
- Once we randomly return to a previously used internal state, we enter a cycle.
- This happens quite quickly! For a block size of $64$ bits:
  - Only $2^{32}$ blocks until we expect it to occur.
  - $2^{21}$ blocks until the probability is more than $2^{-20}$ that this has occurred.

Subsection 3

# ANSI X9.31-1998 A.2.4 DRBG

# A Somewhat Better Idea

ANSI X9.31-1998 A.2.4



# A Somewhat Better Idea: Notes

- Seed by selecting a key, $*K$, and a one block $V$.
- The updating $DT$ field helps prevent cycles.
- We don't directly expose the internal state.
- We never update $*K$ (until we rekey).
- We can't gracefully provide additional entropy.
- The internal state size is still quite small, and can't be expanded.
- Seeds can only be as large as the internal state, so must be full entropy to obtain a reasonable security level.

## Subsection 4

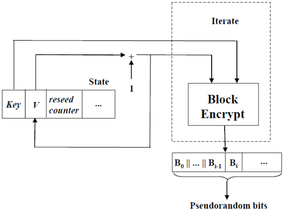
# CTR-DRBG

# A Very Good Design: Generate

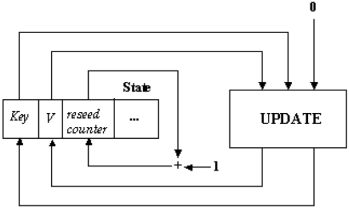
## Stages 2 and 3 of NIST’s CTR-DRBG *Generate*:

(Stage 1 is not directly relevant to this discussion.)

Stage 2:

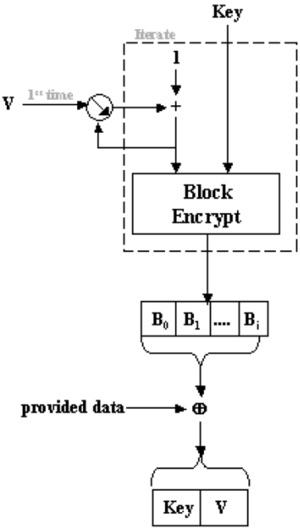


Stage 3:



Source: NIST SP800-90A

# A Very Good Design: Update



Source: NIST SP800-90A

UNIVERSITY of CALIFORNIA · IRVINE

# A Very Good Design: Notes

- This design allows for effectively arbitrary length seed input.
- Seeding input produces $V$ and $Key$.
- $V$ is one cipher block long
- $Key$ and $V$ are updated during the Instantiate, Reseed, Generate operations.
- $Key$ and $V$ are segregated for the generation loop (reducing the likelihood of a cycle).
- Update mixes $Key$ and $V$ (updating all the state between Generates).
- Uses block cipher in a Counter-like mode to produce output bits and mix $Key$ and $V$.
- Very unlikely to enter a cycle (with probability less than $2^{-40}$ when used as directed).

## Section 4

# Conclusion

# Conclusion

- For reasonable security, it is necessary to use both a DBRG and a NDRBG.
- For the NDRBG
  - Only use sources that you *really* understand.
  - Try to be very conservative with entropy analysis.
  - Test!
- For the DRBG, use a well understood and evaluated design. The design should:
  - be based on a well understood cryptographic primitive.
  - allow for large seed input.
  - allow for periodic reseeding.
  - not keep any state data fixed.
  - never discard data that might contain entropy.
  - not be susceptible to cycles.