Randomness: theory and practice

Alexander Shen

Yurifest: May 2020

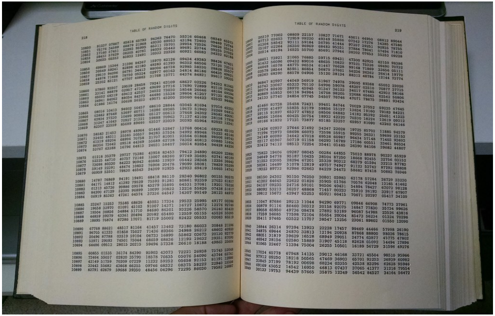
paradox of individual random objects

- fair coin assumption says that all sequences of N bits are equiprobable as outcomes of fair coin tossing
- still some of them refute the fair coin model while other (“random bit sequences”) do not

Is randomness real?

randomness around us

more serious efforts

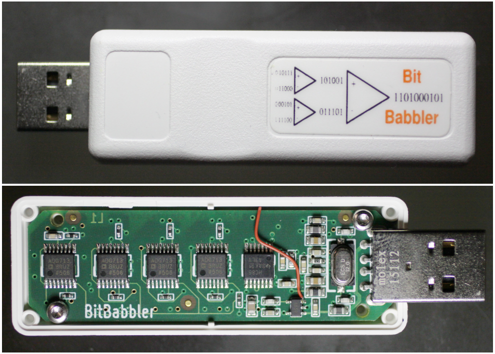


2 TABLE OF RANDOM DIGITS

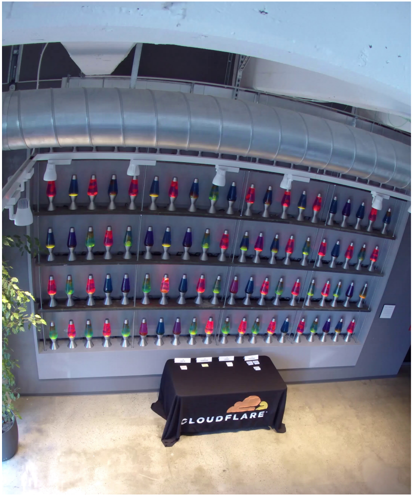
00050	09188	20097	32825	39527	04220	86304	83389	87374	64278	58044
00051	90045	85497	51981	50654	94938	81997	91870	76150	68476	64659
00052	73189	50207	47677	26269	62290	64464	27124	67018	41361	82760
00053	75768	76490	20971	87749	90429	12272	95375	05871	93823	43178
00054	54016	44056	66281	31003	00682	27398	20714	53295	07706	17813

Rand Corporation, *A Million Random Digits with 100,000 Normal Deviates* (1955)

electronic devices



...could be exotic



I: probability theory

- test: a set of $T \subset \{0,1\}^N$ that has very small probability
- if $x \in A$, then x fails the test
- large deviations theorems
- limit theorems
- statistics (χ^2, Kolmogorov–Smirnov, ...)

Statistics as a proof

- “test should be fixed before the experiment”: unclear but essential
- Bonferroni correction
- problems: how to choose p-value?
- what tests are acceptable?
- Gurevich, Passmore: Impugning randomness, convincingly (Studia Logica 2012)

II: algorithmic information theory

- randomness ≈ incompressibility
- no program shorter than the sequence can produce it
- Kolmogorov complexity ≈ length
- obstacle I: non-computability of complexity (one can prove non-randomness but not randomness)
- obstacle II: arbitrary constants
- still the choice of programming language in advance is more reasonable than the choice of the test

III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)
- G: short n-bit seed $\longmapsto$ long N-bit sequence
- mapping G easy to compute (all images compressible)
- no easily computable test $T \subset \{0,1\}^N$ can distinguish the output from random N bits:

$$\Pr_{x \in \{0,1\}^n}[G(x) \in T] \approx \Pr_{y \in \{0,1\}^N}[y \in T]$$

- easily computable $\approx$ polynomial-size circuits
- exist iff one-way functions exist (Hastad, Impagliazzo, Luby, Levin)

IV: combinatorics, randomness extractors

- $D\colon \mathbb{B}^n \times \mathbb{B}^d \longrightarrow \mathbb{B}^m$:
  $D(\text{reasonable random long}, \text{short independent random})$
  almost random and rather long
- if ξ is a random variable in $\mathbb{B}^n$ with large min-entropy, ρ is an independent uniform random variable in $\mathbb{B}^d$, then $D(\xi,\rho)$ has distribution that is statistically (L_1) close to the uniform on $\mathbb{B}^m$
- existence can be proven
- some explicit constructions
- also two independent weakly random sources

random bits

needed for:

- random sampling in statistics
- draws, lotteries,...
- Monte-Carlo computations
- more general, simulations
- randomized algorithms could be more efficient:
  - quick sort with random pivot
  - primality testing
  - computing an average of some array
- cryptographic protocols (one-time pad, secret sharing)

“deterministic random bits”

- fix $f\colon \mathbb{B}^n \longrightarrow \mathbb{B}^n$, let $x_{n+1} = f(x_n)$
- von Neumann: middle digits of a square
- linear/affine mapping in a finite field
- not random in any reasonable sense (computable, predictable)
- but still could have good convergence for Monte-Carlo etc.

hardware randomness

- also called “non-deterministic random generators”
- some process (thermal noise, radioactive decay, photons reflection, environment, ...) is used
- physics claims some probability distribution
- usually some conditioning/whitening is needed
- “centaurs”: hardware seed generation plus deterministic transformation (Yao, Blum–Micali)
- a special type of “whitening”: no hope to get uniform randomness, just computably indistinguishable

what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis H_0 = uniform distribution
- test: a small set of binary strings
- its elements fail the test
- should be specified in advance...
- or be so simple that it could be specified in advance
- “deterministic RNG” may also pass some tests
- conjecture: digits of π form a normal sequence

history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- χ^2-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
- Marsaglia `diehard` (1985–1995): still used
- Brown `dieharder` (2005): more flexible
- NIST 800-22 (2000, 2010), STS
- Simard, l'Ecuyer `TestU01` (2007)

example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- p-values: let $S\colon \mathbb{B}^n \longrightarrow \mathbb{R}$ be any function
- for each $x \in \mathbb{B}^n$ we compute the p-value for x
  $p_S(x) = \Pr[S(r) \geqslant S(x)]$ for random $r \in \mathbb{B}^n$
- p-values for ordered sets (Gurevich, Vovk, 2017)

Secondary tests

- if $p_S(x)$ is very small, x fails the S-test
- if each value of S has negligible probability, $p_S(x)$ is uniformly distributed in $[0,1]$
- so one can use tests (e.g., Kolmogorov–Smirnov) for independent values of $p_S(x)$
- mentioned by Knuth (1969, Art of Programming, v.2)
- widely used in diehard

tests in algorithmic information theory

- Martin-Löf: randomness for infinite sequences
- test: decreasing sequence of open sets (elements of U_i have randomness deficiency $\leqslant i$: $\Pr[U_i] \leqslant 2^{-i}$)
- probability-bounded and expectation-bounded tests (Levin, Gács)
- universal test: finite for random sequences; adding a long prefix of zeros increases deficiency but it remains finite
- Schnorr–Levin–Gács theorem: expression for the universal test in terms of Kolmogorov complexity
- quantitative algorithmic randomness theory

theory vs. practice: ID Quantique

However, if one were to be given a number, it is simply impossible to verify whether it was produced by a random number generator or not. It is hence absolutely essential to consider sequences of numbers in order to study the randomness of the output of such a generator.

It is quite straightforward to define whether a sequence of infinite length is random or not. This sequence is random if the quantity of information it contains – in the sense of Shannon's information theory – is also infinite.

In other words, it must not be possible for a computer program, whose length is finite, to produce this sequence. Interestingly, an infinite random sequence contains all possible finite sequences.

(white paper)

- randomness is mixed with non-computability
- the last statement is false

theory vs. practice: NIST 800-22-1a

- type I error probability of failing the test assuming the null hypothesis H_0 (ok)
- "Type II error probability is $\langle \dots \rangle$ $P\big(\text{accept } H_0 | H_0 \text{ is false}\big)$" (1-4)
- but "H_0 is false" does not define any distribution
- "Unlike α [the probability of Type I error], β is not a fixed value. $\langle \dots \rangle$ The calculation of Type II error β is more difficult than the calculation of α because of the many possible types of non-randomness"
- "If a *P-value* for a test is determined to be equal to 1, then the sequence appears to have perfect randomness" (1-4)
- "For a *P-value* $\geqslant 0.001$, a sequence would be considered to be random with a confidence of 99.9%. For a *P-value* < 0.001, a sequence would be considered to be non-random with a confidence of 99.9%" (1-4)
- two incorrect tests deleted from the second version

theory vs. practice: `diehard[er]`

- passing the test guarantees nothing (ok, unavoidable)
- what about failing the test?
- computation of p-values based on heuristic assumptions
- diehard: secondary tests based on incorrect assumptions
- dieharder: “At this point I think there is rock solid evidence that this test [one of the `diehard` tests] is completely useless in every sense of the word. It is broken, and it is so broken that there is no point in trying to fix it. The problem is that the transformation above is not linear, and doesn’t work. Don’t use it.”

theory vs. practice: entropy

- entropy of a distribution (Shannon)
- for individual objects: Kolmogorov complexity
- a liquid produced by generators and accumulated in pools? "The central mathematical concept underlying this [NIST] Recommendation is entropy. Entropy is defined relative to one's knowledge of an experiment's output prior to observation, and reflects the uncertainty associated with predicting its value – the larger the amount of entropy, the greater the uncertainty in predicting the value of an observation"
- "Each bit of a bitstring with full entropy has a uniform distribution and is independent of every other bit of that bitstring. Simplistically, this means that a bitstring has full entropy if every bit of the bitstring has one bit of entropy; the amount of entropy in the bitstring is equal to its length' (same NIST document)

theory vs. practice: whitening

- Santha–Vazirani sources: $X_1, \ldots, X_n$
- $\Pr[X_i = 1 \mid X_0 = x_0, \ldots, X_{i-1} = x_{i-1}] \in (1/3, 2/3)$
- “no value can be predicted for sure”
- F: a deterministic transformation
- can we guarantee that $F(X_1, \ldots, X_n)$ is close to a fair coin?
- nothing better than $(1/3, 2/3)$
- similar results for k bits: for $F\colon \mathbb{B}^n \to \mathbb{B}^k$ there is SV source and some k-bit output string that appear with probability at least $(2/3)^k$ instead of $(1/2)^k$

theory vs. practice: randomness extraction

- $F(X,R)$ is statistically close to uniform randomness if
  - X is long and has reasonable min-entropy
  - R is short but perfectly random
  - X and R are independent
- IDquantique uses this approach
- but for fixed R (generated, sent with the device)
- so nothing is guaranteed
- strong extractor: $(F(X,R),R) \approx$ uniform
- can be saved, but only with half of the security parameter

theory vs. practice: using independence

- randomness extractors with several independent sources
- exist with good parameters
- only the simplest approach seems to be used
- if $X_1,\ldots,X_n$ are independent and $\Pr[X_i=1]\in(1/3,2/3)$, $X_1\oplus\ldots\oplus X_n$ is exponentially close to a fair coin
- independence is physically plausible

theory vs. practice: coding

- dieharder: non-reproducible results even with fixed seed
- wrong computation of Kolmogorov−Smirnov statistics
- tests are hard to debug
- NIST says:
  In practice, many reasons can be given to explain why a data set has failed a statistical test. The following is a list of possible explanations. The list was compiled based upon NIST statistical testing efforts.
  1. An incorrectly programmed statistical test.
  2. An underdeveloped (immature) statistical test.
  3. An improper implementation of a random number generator.
  4. Improperly written codes to harness test input data.
  5. Poor mathematical routines for computing *P-values*.
  6. Incorrect choices for input parameters.

how to make tests robust

- we do not know the exact distribution of a statistic S and p-values are unreliable
- for secondary test it is not necessary if we use Kolmogorov–Smirnov test for two samples: $S(x_1),\ldots,S(x_n)$ and $S(y_1),\ldots,S(y_m)$
- $x_1,\ldots,x_n$ from the generator we test, $y_1,\ldots,y_m$ from a reference generator
- may reject a good generator using a bad reference
- $S(x_1),\ldots,S(x_n)$ vs $S(x_{n+1}\oplus y_1),\ldots,S(x_{n+m}\oplus y_m)$

survey of available generators

parameters to take into account:

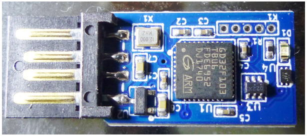
- noise source
- whitening
- access to raw noise
- rate
- cost
- software integration
- bonus: open source hard/software

Araneus

$$$, zener noise, 100 kbits/s, raw=no, whitening=?

"The raw output bits from the A/D converter are then further processed by an embedded microprocessor to combine the entropy from multiple samples into each final output bit, resulting in a random bit stream that is practically free from bias and correlation"

Gniibe

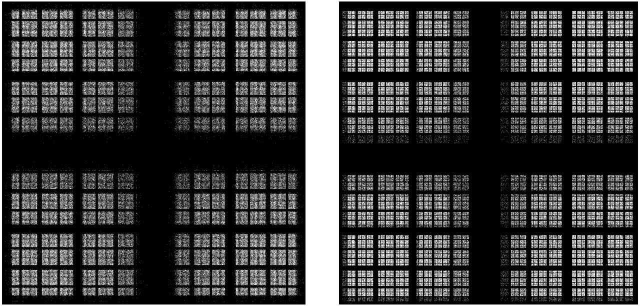



$$, environment noise, 3 mbits/s, access to raw bits, open source (based on GNU microprocesssor unit), whitening=CRC32 + SHA-256

Infinite Noise

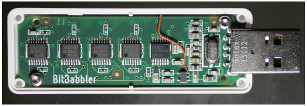
$$, electronic noise, $x \mapsto 2x - 1$ digitization, 300 kbits/s, access to raw bits, whitening=SHA3

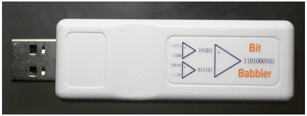
analysis of raw noise bits



infinite noise: measured vs. model

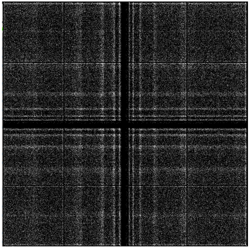
Bitbabbler



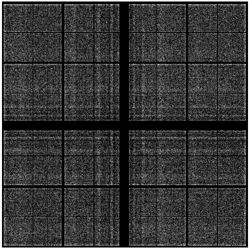


\$\$–\$\$\$, electronic noise, $x \longmapsto 2x - 1$ digitization, 2.5 mbits/s default, 4 independent generators (\$150 version), access to raw bits, variable discretization rate, whitening=XOR

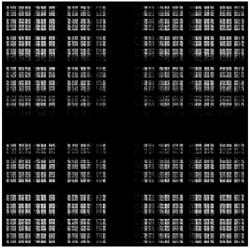
Bitbabbler: changing rate



100 kHz

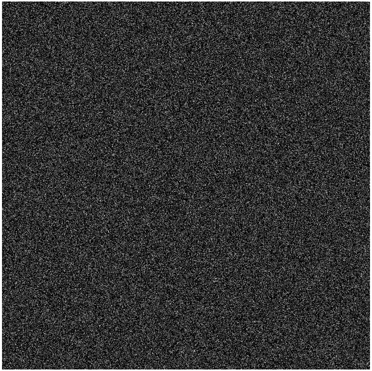


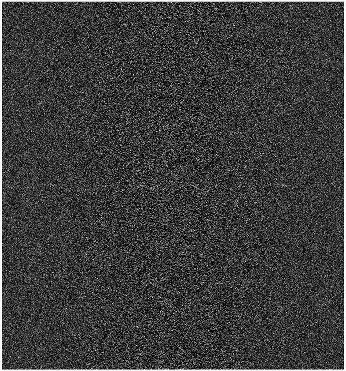
default rate 2.5 MHz



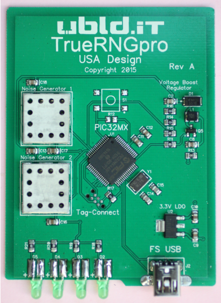
5 MHz

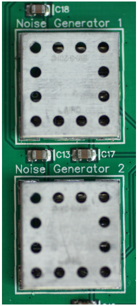
2 or3 XOR





TrueRNG



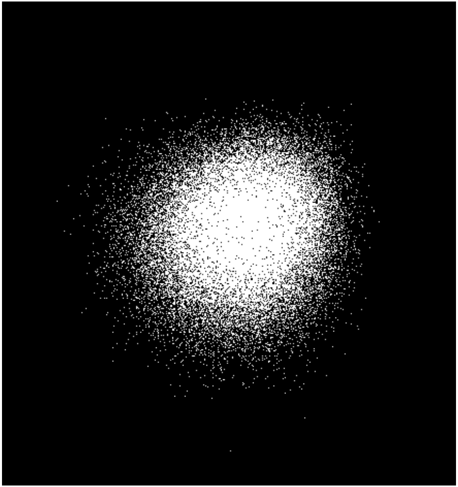


$$–$$$, zener noise + ADC,

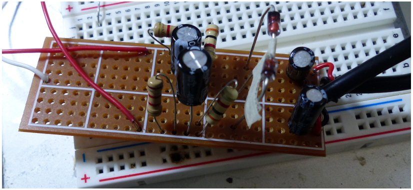
3.2 mbits/s, 2 independent generators ($100 version),

access to raw bits, whitening=XOR/CRC

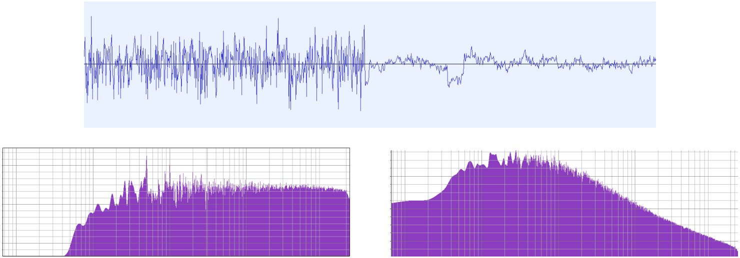
TrueRNG raw noise



DIY approach



DIY: not all noise sources are the same

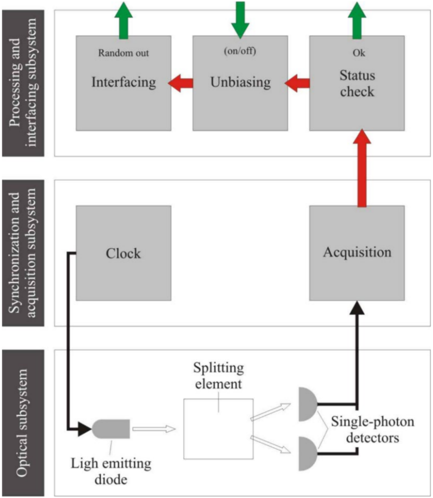


two zener diodes from the same roll

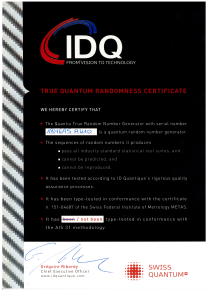
ID Quantique

$$$–$$$$, photon detectors, 4 mbits/s, no access to raw bits, whitening?, additional randomness extraction available

ID Quantique: scheme



certificates as randomness theater?


IDQ
FROM VISION TO TECHNOLOGY

TRUE QUANTUM RANDOMNESS CERTIFICATE

WE HEREBY CERTIFY THAT

- The Quantis True Random Number Generator with serial number 184695 A110 is a quantum random number generator.
- The sequences of random numbers it produces
  - pass all industry standard statistical test suites, and
  - cannot be predicted, and
  - cannot be reproduced.
- It has been tested according to ID Quantique's rigorous quality assurance processes.
- It has been type-tested in conformance with the certificate n. 151-04687 of the Swiss Federal Institute of Metrology METAS.
- It has ~~been~~ / not been type-tested in conformance with the AIS 31 methodology.

Grégoire Ribordy
Chief Executive Officer
www.idquantique.com

SWISS QUANTUM

still fails dieharder/ent tests (before optional randomness extractor)

security through obscurity

- NIST recommends (and insists) on using cryptographic whitening
- “approved hash function”
- nothing is proven about them
- and even it were, it won’t help

NIST says:

> *Hash_DRBG's [the random generator based on hash functions] security depends on the underlying hash function's behavior when processing a series of sequential input blocks. If the hash function is replaced by a random oracle, Hash_DRBG is secure. It is difficult to relate the properties of the hash function required by Hash_DRBG with common properties, such as collision resistance, pre-image resistance, or pseudorandomness.*

vulnerabilities

- software attack if a microprocessor is used
- undetected failure of noise source
- whitening obscures failures
- obscure hash function as a Troyan horse
- distribution close to random but still distinguishable
- last but not least: stupid errors (e.g., AMD Zen FF random generator)

remedies

- `xor` of independent devices
- possible to make in-house
- open source hardware/software
- several reasonably cheap commercial generators, no need for a fancy one

Happy Birthday to Ю!