random bits in practice and theory

RaCAF project
random bits in practice and theory
random objects?

paradox of individual random objects
random bits in practice and theory
random objects?

paradox of individual random objects

- fair coin assumption says that all sequences of $N$ bits are equiprobable as outcomes of fair coin tossing
random bits in practice and theory
random objects?

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
random bits in practice and theory

random objects?

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?
random bits in practice and theory

random objects?

randomness around us
random bits in practice and theory

random objects?

more serious efforts

Rand Corporation, *A Million Random Digits with 100,000 Normal Deviates* (1955)
random bits in practice and theory
random objects?

electronic devices
I: probability theory
test: a set of $T \subset \{0, 1\}^N$ that has very small probability
I: probability theory

- test: a set of $T \subseteq \{0, 1\}^N$ that has very small probability
- if $x \in A$, then $x$ fails the test
I: probability theory

- test: a set of $T \subseteq \{0, 1\}^N$ that has very small probability
- if $x \in A$, then $x$ fails the test
- large deviations theorems
I: probability theory

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

I: probability theory
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 ($\chi^2$, Kolmogorov–Smirnov, ...)
- “test should be fixed before the experiment”: unclear but essential
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 ($\chi^2$, Kolmogorov–Smirnov, …)
- “test should be fixed before the experiment”: unclear but essential
- Bonferroni correction
II: algorithmic information theory
II: algorithmic information theory

- randomness ⇐ incompressibility
II: algorithmic information theory

- randomness $\approx$ incompressibility
- no program shorter than the sequence can produce it
II: algorithmic information theory

- randomness $\approx$ incompressibility
- no program shorter than the sequence can produce it
- Kolmogorov complexity $\approx$ length
II: algorithmic information theory

- randomness $\approx$ incompressibility
- no program shorter than the sequence can produce it
- Kolmogorov complexity $\approx$ length
- obstacle I: non-computability of complexity (one can prove non-randomness but not randomness)
II: algorithmic information theory

- randomness $\approx$ incompressibility
- no program shorter than the sequence can produce it
- Kolmogorov complexity $\approx$ length
- obstacle I: non-computability of complexity (one can prove non-randomness but not randomness)
- obstacle II: arbitrary constants
II: algorithmic information theory

- randomness $\approx$ incompressibility
- no program shorter than the sequence can produce it
- Kolmogorov complexity $\approx$ 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
III: computational complexity

▶ not individual sequences but mappings (Yao, Blum–Micali)
III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)

\[ G: \text{short } n\text{-bit seed} \mapsto \text{long } N\text{-bit sequence} \]
III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)
- \( G: \text{short } n\text{-bit seed} \mapsto \text{long } N\text{-bit sequence} \)
- mapping \( G \) easy to compute
III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)
- $G$: short $n$-bit seed $\mapsto$ long $N$-bit sequence
- mapping $G$ easy to compute (all images compressible)
III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)
- $G$: short $n$-bit seed $\mapsto$ 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:

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

- not individual sequences but mappings (Yao, Blum–Micali)
- \( G: \) short \( n \)-bit seed \( \mapsto \) 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
III: computational complexity

- not individual sequences but mappings (Yao, Blum–Micali)
- $G$: short $n$-bit seed $\mapsto$ 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
IV: combinatorics, randomness extractors

$D : \mathbb{B}^n \times \mathbb{B}^d \rightarrow \mathbb{B}^m$:

$D(\text{reasonable random long, short independent random})$

almost random and rather long
IV: combinatorics, randomness extractors

- $D: \mathbb{B}^n \times \mathbb{B}^d \rightarrow \mathbb{B}^m$:
  
  $D(\text{reasonable random long, short independent random})$
  
  almost random and rather long

- if $\xi$ is a random variable in $\mathbb{B}^n$ with large min-entropy, $\rho$ 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 $\mathbb{B}^m$
IV: combinatorics, randomness extractors

- $D : \mathbb{B}^n \times \mathbb{B}^d \rightarrow \mathbb{B}^m$:
  - $D$ (reasonable random long, short independent random) almost random and rather long

- If $\xi$ is a random variable in $\mathbb{B}^n$ with large min-entropy, $\rho$ 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
IV: combinatorics, randomness extractors

- $D: \mathbb{B}^n \times \mathbb{B}^d \rightarrow \mathbb{B}^m$:
  $D$ (reasonable random long, short independent random) almost random and rather long

- if $\xi$ is a random variable in $\mathbb{B}^n$ with large min-entropy, $\rho$ 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
IV: combinatorics, randomness extractors

- $D : \mathbb{B}^n \times \mathbb{B}^d \rightarrow \mathbb{B}^m$:
  $D$ (reasonable random long, short independent random) almost random and rather long

- If $\xi$ is a random variable in $\mathbb{B}^n$ with large min-entropy, $\rho$ 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 in practice and theory
— randomness generators

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)
random bits

needed for:

▶ random sampling in statistics
random bits

needed for:

- random sampling in statistics
- draws, lotteries,...
random bits

needed for:

- random sampling in statistics
- draws, lotteries,...
- Monte-Carlo computations
random bits

needed for:

- random sampling in statistics
- draws, lotteries,…
- Monte-Carlo computations
- more general, simulations
random bits

needed for:

- random sampling in statistics
- draws, lotteries, ...
- Monte-Carlo computations
- more general, simulations
- randomized algorithms could be more efficient:
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
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
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
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”
“deterministic random bits”

▶ fix $f: \mathbb{B}^n \rightarrow \mathbb{B}^n$, let $x_{n+1} = f(x_n)$
“deterministic random bits”

- fix \( f: \mathbb{B}^n \rightarrow \mathbb{B}^n \), let \( x_{n+1} = f(x_n) \)
- von Neumann: middle digits of a square
“deterministic random bits”

- fix $f: \mathbb{B}^n \rightarrow \mathbb{B}^n$, let $x_{n+1} = f(x_n)$
- von Neumann: middle digits of a square
- linear/affine mapping in a finite field
“deterministic random bits”

- fix $f: \mathbb{B}^n \rightarrow \mathbb{B}^n$, 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)
“deterministic random bits”

- fix $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$, 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.
random bits in practice and theory

Randomness generators

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
hardware randomness

- also called “non-deterministic random generators”
hardware randomness

- also called “non-deterministic random generators”
- some process (thermal noise, radioactive decay, photons reflection, environment, ...) is used
random bits in practice and theory

randomness generators

hardware randomness

▶ also called “non-deterministic random generators”
▶ some process (thermal noise, radioactive decay, photons reflection, environment, ...) is used
▶ physics claims some probability distribution
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
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)
randomness generators

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?
what is a test?

- hardware RNG: special case of statistical testing
what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
- test: a small set of binary strings
what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
- test: a small set of binary strings
- its elements fail the test
what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
- test: a small set of binary strings
- its elements fail the test
- should be specified in advance...
random bits in practice and theory

randomness tests

what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
- 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
what is a test?

- hardware RNG: special case of statistical testing
- null hypothesis $H_0 = \text{uniform distribution}$
- 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
What is a test?

- Hardware RNG: special case of statistical testing
- Null hypothesis $H_0 = \text{uniform distribution}$
- 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 $\pi$ form a normal sequence
history of tests
history of tests

▶ early history described in Knuth (vol.2, 1969)
random bits in practice and theory
randomness tests

history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers (#0 ≈ #1)
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
- Brown dieharder (2005): more flexible
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
- Brown dieharder (2005): more flexible
- NIST 800-22 (2000, 2010), STS
history of tests

- early history described in Knuth (vol.2, 1969)
- law of large numbers ($\#0 \approx \#1$)
- $\chi^2$-tests for frequencies of bytes, etc.
- used when generating tables of random numbers
- Brown dieharder (2005): more flexible
- NIST 800-22 (2000, 2010), STS
- Simard, l’Ecuyer TestU01 (2007)
random bits in practice and theory
randomness tests

example of tests
example of tests

▶ incompressibility (gzip as a test)
random bits in practice and theory

randomness tests

example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory

$p$-values: let $S : \mathbb{B}^n \to \mathbb{R}$ be any function

- for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$

$$p_S(x) = \Pr[S(r) \geq S(x)]$$

for random $r \in \mathbb{B}^n$

- 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 $p_S(x)$

- secondary tests (in Knuth, widely used in diehard)
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- *p*-values: let $S: \mathbb{B}^n \rightarrow \mathbb{R}$ be any function
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- $p$-values: let $S : \mathbb{B}^n \rightarrow \mathbb{R}$ be any function
- for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- $p$-values: let $S: \mathbb{B}^n \rightarrow \mathbb{R}$ be any function
- for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$
  \[ p_S(x) = \Pr[S(r) \geq S(x)] \] for random $r \in \mathbb{B}^n$
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- $p$-values: let $S : \mathbb{B}^n \to \mathbb{R}$ be any function
- for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$
  \[ p_S(x) = \Pr[S(r) \geq S(x)] \] for random $r \in \mathbb{B}^n$
- if $p_S(x)$ is very small, $x$ fails the $S$-test
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- \( p \)-values: let \( S : \mathbb{B}^n \rightarrow \mathbb{R} \) be any function
  
  for each \( x \in \mathbb{B}^n \) we compute the \( p \)-value for \( x \)
  
  \[ p_S(x) = \Pr[S(r) \geq S(x)] \text{ for random } r \in \mathbb{B}^n \]
  
  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 \([0, 1]\)
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- $p$-values: let $S : \mathbb{B}^n \to \mathbb{R}$ be any function
  - for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$
    $$p_S(x) = \Pr[S(r) \geq S(x)]$$
    for random $r \in \mathbb{B}^n$
  - 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 $p_S(x)$
example of tests

- incompressibility (gzip as a test)
- limit theorems in probability theory
- $p$-values: let $S: \mathbb{B}^n \to \mathbb{R}$ be any function
- for each $x \in \mathbb{B}^n$ we compute the $p$-value for $x$
  $$p_S(x) = \Pr[S(r) \geq S(x)]$$
  for random $r \in \mathbb{B}^n$
- 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 $p_S(x)$
- secondary tests (in Knuth, widely used in diehard)
tests in algorithmic information theory
tests in algorithmic information theory

▶ Martin-Löf: randomness for infinite sequences
tests in algorithmic information theory

- Martin-Löf: randomness for infinite sequences
- test: decreasing sequence of open sets (elements of $U_i$ have randomness deficiency $\leq i$: $\Pr[U_i] \leq 2^{-i}$)
tests in algorithmic information theory

- Martin-Löf: randomness for infinite sequences
- test: decreasing sequence of open sets (elements of $U_i$ have randomness deficiency $\leq i$: $\Pr[U_i] \leq 2^{-i}$)
- probability-bounded and expectation-bounded tests (Levin, Gács)
tests in algorithmic information theory

- Martin-Löf: randomness for infinite sequences
- test: decreasing sequence of open sets (elements of $U_i$ have randomness deficiency $\leq i$: $\Pr[U_i] \leq 2^{-i}$)
- 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
tests in algorithmic information theory

- Martin-Löf: randomness for infinite sequences
- test: decreasing sequence of open sets (elements of $U_i$ have randomness deficiency $\leq i$: $\Pr[U_i] \leq 2^{-i}$)
- 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
random bits in practice and theory
randomness tests

tests in algorithmic information theory

▶ Martin-Löf: randomness for infinite sequences
▶ test: decreasing sequence of open sets (elements of \(U_i\) have randomness deficiency \(\leq i: \Pr[U_i] \leq 2^{-i}\))
▶ 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
goals of RaCAF
goals of RaCAF

▶ try to bridge the gap between theory and practice
goals of RaCAF

- try to bridge the gap between theory and practice
- isolate the problematic points
goals of RaCAF

▶ try to bridge the gap between theory and practice
▶ isolate the problematic points
▶ evaluations/recommendations
goals of RaCAF

- try to bridge the gap between theory and practice
- isolate the problematic points
- evaluations/recommendations
- improvements
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)
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
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
- (making the last statement false)
theory vs. practice: NIST 800-22-1a
theory vs. practice: NIST 800-22-1a

- type I error probability of failing the test assuming the null hypothesis $H_0$ (ok)
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 \( \ldots \) $P(\text{accept } H_0 | H_0 \text{ is false})$” (1-4)
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 $\ldots$ $P(\text{accept } H_0|H_0 \text{ is false})$” (1-4)
- but “$H_0$ is false” does not define any distribution
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 $P(\text{accept } H_0|H_0 \text{ is false})$” (1-4)
- but “$H_0$ is false” does not define any distribution
- “Unlike $\alpha$ [the probability of Type I error], $\beta$ is not a fixed value. $\langle \ldots \rangle$ The calculation of Type II error $\beta$ is more difficult than the calculation of $\alpha$ because of the many possible types of non-randomness”
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 $\ldots P(\text{accept } H_0 | H_0 \text{ is false})$” (1-4)
- but “$H_0$ is false” does not define any distribution
- “Unlike $\alpha$ [the probability of Type I error], $\beta$ is not a fixed value. $\ldots$ The calculation of Type II error $\beta$ is more difficult than the calculation of $\alpha$ 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)
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 $\mathcal{P}(\text{accept } H_0|H_0 \text{ is false})$” (1-4)
- but “$H_0$ is false” does not define any distribution
- “Unlike $\alpha$ [the probability of Type I error], $\beta$ is not a fixed value. $\langle \ldots \rangle$ The calculation of Type II error $\beta$ is more difficult than the calculation of $\alpha$ 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 $\geq 0.001$, 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)
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 \( \ldots P(\text{accept } H_0|H_0 \text{ is false}) \)” (1-4)
- but “\( H_0 \) is false” does not define any distribution
- “Unlike \( \alpha \) [the probability of Type I error], \( \beta \) is not a fixed value. \( \ldots \) The calculation of Type II error \( \beta \) is more difficult than the calculation of \( \alpha \) because of the many possible types of non-randomness”
- “If a \textit{P-value} for a test is determined to be equal to 1, then the sequence appears to have perfect randomness” (1-4)
- “For a \textit{P-value} \( \geq 0.001 \), a sequence would be considered to be random with a confidence of 99.9%. For a \textit{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]
theory vs. practice: diehard[er]

- passing the test guarantees nothing (ok, unavoidable)
theory vs. practice: diehard[er]

- passing the test guarantees nothing (ok, unavoidable)
- what about failing the test?
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
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
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.”
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)
random bits in practice and theory

theory vs. practice: entropy

- entropy of a distribution (Shannon)
theory vs. practice: entropy

▶ entropy of a distribution (Shannon)
▶ for individual objects: Kolmogorov complexity
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”
random bits in practice and theory

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
theory vs. practice: whitening

- Santha–Vazirani sources: $X_1, \ldots, X_n$
theory vs. practice: whitening

- Santha–Vazirani sources: $X_1, \ldots, X_n$
- $\Pr[X_i = 1 | X_0 = x_0, \ldots, X_{i-1} = x_{i-1}] \in (1/3, 2/3)$
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”
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
theory vs. practice: whitening

- Santha–Vazirani sources: $X_1, \ldots, X_n$
- $\Pr[X_i = 1 | X_0 = x_0, \ldots, X_{i-1} = x_{i-1}] \in (1/3, 2/3)$
- “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?
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 $(1/3, 2/3)$
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 $F: \mathbb{B}^n \rightarrow \mathbb{B}^k$ 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
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: $(F(X, R), R)$ uniform
  - can be saved, but only with half of the security parameter
theory vs. practice: randomness extraction

- $F(X, R)$ is statistically close to uniform randomness if
  - $X$ is long and has reasonable min-entropy
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

IDquantique uses this approach but for fixed $R$ (generated, sent with the device) so nothing is guaranteed.

A strong extractor: $(F(X, R), R)$ is uniform but can be saved, but only with half of the security parameter.
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.

A strong extractor is $(F(X,R), R)$ uniform, but only with half of the security parameter.
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
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)
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
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
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
Random bits in practice and theory

Theory vs. practice: using independence
theory vs. practice: using independence

- randomness extractors with several independent sources
theory vs. practice: using independence

▶ randomness extractors with several independent sources
▶ exist with good parameters
random bits in practice and theory

theory vs. practice: using independence

- randomness extractors with several independent sources
- exist with good parameters
- only the simplest approach seems to be used
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
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
theory vs. practice: coding

- dieharder: non-reproducible results even with fixed seed
theory vs. practice: coding

- dieharder: non-reproducible results even with fixed seed
- wrong computation of Kolmogorov–Smirnov statistics
theory vs. practice: coding

- dieharder: non-reproducible results even with fixed seed
- wrong computation of Kolmogorov–Smirnov statistics
- tests are hard to debug
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
how to make tests robust

- we do not know the exact distribution of a statistic $S$
  and $p$-values are unreliable
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 $S(y_1), \ldots, S(y_m)$
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
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
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:
survey of available generators

parameters to take into account:

- noise source
survey of available generators

parameters to take into account:

▶ noise source
▶ whitening
survey of available generators

parameters to take into account:

▶ noise source
▶ whitening
▶ access to raw noise
survey of available generators

parameters to take into account:

▶ noise source
▶ whitening
▶ access to raw noise
▶ rate
survey of available generators

parameters to take into account:

- noise source
- whitening
- access to raw noise
- rate
- cost
survey of available generators

parameters to take into account:

- noise source
- whitening
- access to raw noise
- rate
- cost
- software integration
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
random bits in practice and theory

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”
random bits in practice and theory

Gniibe

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

\$$, \text{electronic noise, } x \mapsto 2x - 1 \text{ digitization, } 300 \text{ kbits/s, access to raw bits, whitening=SHA3} \$$
random bits in practice and theory

RaCAF

analysis of raw noise bits

infinite noise: measured vs. model
Bitbabbler

\$\$–\$\$$, electronic noise, \( x \mapsto 2x - 1 \) 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 or 3 XOR
TrueRNG

$$-$$, zener noise + ADC,
3.2 mbits/s, 2 independent generators ($100 version),
access to raw bits, whitening=XOR/CRC
TrueRNG raw noise
random bits in practice and theory

— RaCAF

DIY approach

![Image 1](image1.png)
![Image 2](image2.png)
DIY: not all noise sources are the same

two zener diodes from the same roll
random bits in practice and theory

ID Quantique

$$\$\$–\$$\$$\$$, photon detectors, 4 mbits/s, no access to raw bits, whitening?, additional randomness extraction available
random bits in practice and theory

ID Quantique: scheme
random bits in practice and theory

paranoid mode on

certificates as randomness theater?

still fails dieharder/ent tests (before optional randomness extractor)
random bits in practice and theory
paranoid mode on

security through obscurity
security through obscurity

- NIST recommends (and insists) on using cryptographic whitening
security through obscurity

- NIST recommends (and insists) on using cryptographic whitening
- “approved hash function”
security through obscurity

- NIST recommends (and insists) on using cryptographic whitening
- "approved hash function"
- nothing is proven about them
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.
random bits in practice and theory
paranoid mode on

vulnerabilities
random bits in practice and theory

paranoid mode on

vulnerabilities

- software attack if a microprocessor is used
vulnerabilities

- software attack if a microprocessor is used
- undetected failure of noise source
vulnerabilities

- software attack if a microprocessor is used
- undetected failure of noise source
- whitening obscures failures
vulnerabilities

- software attack if a microprocessor is used
- undetected failure of noise source
- whitening obscures failures
- obscure hash function as a Trojan horse
vulnerabilities

▶ software attack if a microprocessor is used
▶ undetected failure of noise source
▶ whitening obscures failures
▶ obscure hash function as a Trojan horse
▶ distribution close to random but still distinguishable
vulnerabilities

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

remedies
remedies

▶ xor of independent devices
remedies

- XOR of independent devices
- Possible to make in-house
remedies

- XOR of independent devices
- Possible to make in-house
- Open source hardware/software
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
remedies

▷ $\text{xor}$ of independent devices
▷ possible to make in-house
▷ open source hardware/software
▷ several reasonably cheap commercial generators, no need for a fancy one

THANKS!