True random number generation

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

– John von Neumann (1951)
True random numbers

Not Pseudo-Random Number Generator
• PRNG is algorithm operating on some seed
• TRNG measures physical state of random system

Usual procedure
• measurement of truly random system
• apply algorithm improving uniformity
• optional: then use as seed in PRNG
Sources of randomness

• noisy resistors
• ring oscillators
• avalanche diodes
• metastable flipflops
• antenna noise
• acoustic noise
• nuclear decay
• unstable lasers
• ...

...
Odd number of inverters in circular configuration

- conflicting logical state
- oscillation between 0/1 state
- propagation time partly random
- exact timing very sensitive to thermal noise
  - ‘jitter’
Extracting randomness from jitter
• XOR several oscillators (analog operation)
• sample the analog signal
• force to logical 0 or 1
  - exact time of flank is Gaussian-distributed
• fill rate \( f \): fraction of time where signal is unpredictable
  - can be tuned by choosing \( k, l \)
Equivalent circuit for resistor:

\[ \langle V^2 \rangle = 4kT R \Delta f \]

Amplitude has Gaussian distribution

\[ \langle V^2 \rangle = 4kT R \Delta f \]

- **k** = Boltzmann constant
- **T** = temperature (Kelvin)
- **R** = resistance
- **\Delta f** = measured range of frequencies

*Johnson & Niquist, 1928*
The Intel RNG

Component of the Intel 80802 chip

*improved version of von Neumann algorithm*

*variable bit rate*

*75 Kbit/s after post-processing*
• Unstable atomic nucleus
• Exact moment of decay unpredictable
• \( \lambda = \) prob of decay per time unit
• \( \Pr[\text{nucleus still exists}] = \exp(-\lambda t). \)
• Very tamper-proof!

Start with \( N \) nuclei; Count \#clicks in time \( \Delta t \).

\[
\Pr[\text{\#clicks is } k] = e^{-N\lambda \Delta t} \frac{(N\lambda \Delta t)^k}{k!}
\]

\( \text{Poisson distribution} \)

Radio-active decay

- Unstable atomic nucleus
- Exact moment of decay unpredictable
- \( \lambda = \) prob of decay per time unit
- \( \Pr[\text{nucleus still exists}] = \exp(-\lambda t). \)
- Very tamper-proof!
Algorithms for randomness extraction

**Known continuous distribution \( f(x) \)**
- generic procedure
- uses cumulative distr. function (cdf)

**Known discrete distribution**
- cdf + binning
- von Neumann algorithm
- piling it up: XOR-ing bits together
- resilient functions

**Unknown discrete distribution**
- universal hash functions
- q-wise independent hashing
Known continuous distribution
Continuous random variables

- $X \sim f$
- Cumulative distribution function $F$
  - $\text{Prob}[X<x] = F(x)$.
- The variable $Y := F(X)$ is uniform!

Figure ysv: The probability is identical to $g(y)$, with $dy = f(x)dx$.

Figure ysw:
(a) $N$ intervals of equal area (i.e. probability) under the Gaussian curve $f(x)$, for $N=8$. The boundary $x_i$ between regions $i$ and $i'$ lies at $F(x_i) = i/N$.
(b) The corresponding picture, but now for the variable $Y = F(X) \in [u,v]$. The pdf of $Y$ is uniform, and the equiprobable intervals all have equal width $v/N$.

3.3.2 Known discrete source; difference with the continuum case
Consider a discrete pmf $\{p_i\}_{i=1}^n$ for some RV $X \in \mathbb{R}$ which can take values $\{x_i\}_{i=1}^n$. You could try to represent the pmf as a fake pdf $f_{\text{mp}}(x) = \sum_{i=1}^n p_i f(x_i)$, and then apply Theorem ysw to obtaining $Y = F_{\text{mp}} X$ with $g(y) = \sum_{i=1}^n p_i F(x_i)$. Does this yield a uniform $Y$? No! The delta functions in $f_{\text{mp}}$ are not the smeared-out functions before the limit but after the limits. The source $X$ cannot really take values other than $x_i$. We end up with a discrete RV $Y$ with the following pmf.

Equiprobable intervals
Known discrete distribution
Let’s try the cdf trick

\[ f(x) = \sum_{i=1}^{n} p_i \delta(x - x_i) \]

\[ Y = \sum_{i=1}^{n} p_i \Theta(X - x_i) \]

Close to cdf of uniform distribution

not perfect, but it may do
von Neumann algorithm

Source = stream of bits from biased coin.
How to remove the bias?

Look at input pairs \((b_1, b_2)\)

\[
\begin{array}{c|c|c}
  b & b & output \\
  \hline
  0 & 0 & -- \\
  0 & 1 & 0 \\
  1 & 0 & 1 \\
  1 & 1 & -- \\
\end{array}
\]

\[b_1 = b_2 : \text{ no output} \]
\[b_1 \neq b_2 : \text{ output } b_1.\]

Question:
1. Why does this work?
2. How much entropy is lost?
von Neumann algorithm: entropy loss

Solution of Exercise 3.3

(a) Let’s say the probability of generating a ‘1’ is \( p \). The events \((b_1, b_2) = (1, 0)\) and \((b_1, b_2) = (0, 1)\) have equal probability \( p \). Other events yield no bit.

(b) Consider \( n \) source bits for some integer \( n \). Then the entropy of the source is \( n h(p) \). The number of bits that is left after application of the von Neumann algorithm is on average \( n/2 \cdot 2p(1-p) \). They are perfectly uniform and carry one bit of entropy each. Thus the entropy loss is \( n [h(p) - p(1-p)] \). The graph below shows the retained fraction \( p(1-p)/h(p) \) as a function of \( p \).

For good sources, \( 3/4 \) of the entropy is thrown away; for very bad sources, almost all entropy.

The entropy of the output of the von Neumann algorithm can also be calculated in a more formal way. Let \( Y_2 \) \{"0", "1"\} be the output when the von Neumann algorithm is applied to two input bits. Here ‘" means: no output. Let \( Z(Y) \) \{"yes", "no"\} be an indicator that says if output is generated. If \( Y = " \) then \( Z = \) no, otherwise \( Z = yes \).

The amount of useful entropy in the output is \( H(Y) - H(Z(Y)) = H(Y|Z) \). The quantity \( H(Y) \) is not the correct thing to look at, because the outcome \( Y = " \) is not useful. We have to subtract \( H(Z) \), the ‘useless’ part of the information.

\[
H(Y|Z = \text{no}) = 0 \quad \text{because knowing that } Z = \text{no} \text{ completely reveals } Y.
\]

\[
H(Y|Z = \text{yes}) = 1 \quad \text{because the output bit, when it exists, is uniform. (See subquestion a.)}
\]

At most \( 1/4 \) of the entropy is kept
Example 3.3
Consider the pmf with $p_1 = p_2 = \frac{1}{4}$ and $p_3 = p_4 = p_5 = p_6 = \frac{1}{8}$. The Shannon entropy is 2.5 bits and the min-entropy is 2 bits. By grouping $p_3, p_4$ together and also $p_5, p_6$, the uniform pmf $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$ is obtained; its entropy is 2 bits.

Example 3.4
Consider the pmf $\{\frac{1}{3}, \frac{1}{6}, \frac{1}{5}, \frac{3}{10}\}$. It has Shannon entropy $\frac{7}{15} \log_2 3 + \frac{1}{2} \log_2 5 + \frac{7}{128}$ bits, and min-entropy $\log_2 3 \frac{1}{58}$ bits. By grouping $\frac{1}{3}, \frac{1}{6} = \frac{1}{2}$ and $\frac{1}{5}, \frac{3}{10} = \frac{1}{2}$ we get a perfectly uniform pmf $\{\frac{1}{2}, \frac{1}{2}\}$ which has 1 bit of entropy. Notice that this is less than the min-entropy.

The min-entropy plays an important role here. If we group probabilities as in the examples above, the number of perfectly uniform bits that we can extract is bounded by the min-entropy. This follows from the fact that the bins cannot be smaller than $p_{\text{max}}$.

Exercise 3.2
What goes wrong when the bins are smaller than $p_{\text{max}}$?

3.3.4 More generic algorithms
The von Neumann algorithm considers a sequence of independent biased coin flips. The von Neumann algorithm removes the bias, but is wasting some of the source's entropy. The algorithm takes a pair of bits $r_1, r_2$ as input and outputs the following:

- if $r_1 = r_2$, no output
- if $r_1 \neq r_2$, output $r_1$

Then the algorithm takes the next two bits of the sequence, etc.

Exercise 3.3
a) Prove that the output of the von Neumann algorithm is uniform, given that the source bits all have the same bias.
b) How much entropy is wasted?

An improvement of the von Neumann algorithm
Several improved de-biasing schemes have been developed. One of them takes four bits as input and outputs either zero, one or two bits. See the table below.

<table>
<thead>
<tr>
<th>input</th>
<th>Neumann</th>
<th>improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>0010</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>0011</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>0101</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>0110</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>01</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1001</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1010</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1110</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1111</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

• generates more bits
• less wasteful

• generates more bits
• less wasteful
$x \in \{0,1\}^n$ containing $t$ 1s

Assign a label $L$ to the permutation that turns $11\cdots 10\cdots 0$ into $x$. $\binom{n}{t}$

$L$ is uniform on $\{0, \cdots, \binom{n}{t} - 1\}$. 
Numerical example:
\[ n = 16 \quad \binom{n}{t} = 4368 = 2^{12} + 272 \]
\[ L \in \{0, \ldots, 4367\} \]

If \( L > 4095 \): No output!
If \( L < 4096 \): Output binary representation of \( L \).

This yields 12 perfect bits with prob. \( 4096/4368 \)
and zero bits with prob. \( 272/4368 \).
→ On average 11.3
Piling up lemma

\[ Y = X_1 \oplus X_2 \oplus \ldots \oplus X_n \]

bias \( \alpha_i = \Pr[X_i=1] - \Pr[X_i=0] \)
\[ |\alpha_i| \leq 1 \]

Combined bias \( \Pr[Y=1] - \Pr[Y=0] = (-1)^{n-1} \prod \alpha_i \)

• reduced bias
• even one occurrence \( \alpha_i=0 \) already gives unbiased \( Y \).
• but ... lots of entropy wasted
Resilient functions

Post-processing

• Apply **resilient** function $\Psi$ insensitive to $k$ bits
• Def. of $[n,m,k]$-resilient:
  ‣ $x \in \{0,1\}^n$, $\Psi(x) \in \{0,1\}^m$. Fix any $k$ bits of $x$.
  ‣ $\text{Prob}[\Psi(X)=y \mid k \text{ bits of } X]$ is **uniform** on $\{0,1\}^m$.
• Can be realized using error-correcting code

Expected:
$k$ out of $n$ bits predictable, but ... we don't know which ones!
Unknown discrete distribution
Definition of a **strong extractor** "Ext" for source min-entropy $m$, length $\ell$ and non-uniformity $\varepsilon$:

- Given a source $X$ with $H_\infty(X) \geq m$
- uniformly drawn public randomness $R$
- $Z = \text{Ext}(X, R) \in \{0,1\}^\ell$.

\[
\mathbb{E}_r \Delta(Z|R = r; U_\ell) \leq \varepsilon
\]

Uniform on $\{0,1\}^\ell$

**In words:**

$\text{Ext}(X,R)$, for known $R$, is $\varepsilon$ away from uniform.
Universal hash functions

Definition:
Universal family of hash functions \{\Phi_r\}

- Functions \(\Phi_r\) from \(\mathcal{X}\) to \(\mathcal{T}\).
- Random seed \(R\), uniformly chosen.
- For any fixed \(x, x'\) with \(x' \neq x\):

\[
\text{Prob}[\Phi_R(x) = \Phi_R(x')] \leq 1/|\mathcal{T}|
\]

Existence of univ. hash functions guarantees existence of strong extractors for certain parameter range!

We will see this in a couple of slides ...
Almost-universal hash functions

Definition 3.11 (Almost universal family of hash functions) Let $\eta \geq 0$ be a constant. Let $\mathcal{R}$, $\mathcal{X}$ and $\mathcal{T}$ be finite sets. Let $\{\Phi_r\}_{r \in \mathcal{R}}$ be a family of hash functions from $\mathcal{X}$ to $\mathcal{T}$. The family $\{\Phi_r\}_{r \in \mathcal{R}}$ is called $\eta$-almost universal iff, for $R$ drawn uniformly from $\mathcal{R}$, it holds that

$$\Pr[\Phi_R(x) = \Phi_R(x')] \leq \eta$$

for all $x, x' \in \mathcal{X}$ with $x' \neq x$.

universal for $\eta = 1/|\mathcal{T}|$
Leftover hash lemma

Theorem 3.12 (Leftover hash lemma) Let \( X \in \mathcal{X} \) be a random variable. Let \( \delta \geq 0 \) be a constant. Let \( F : \mathcal{X} \times \mathcal{R} \rightarrow \{0,1\}^\ell \) be a \( 2^{-\ell}(1 + \delta) \)-almost universal family of hash functions, with seed \( R \in \mathcal{R} \). Then

\[
\Delta(F(X, R); U_\ell R) \leq \frac{1}{2} \sqrt{\delta + 2^{\ell-H_2(X)}},
\]  

(3.17)

Distance of \( F(X, R) \) from uniformity, given \( R \)

Proof is rather long, see appendix in lecture notes.

The \( H_2 \) is the Rényi entropy of order 2,

\[
H_2(X) = - \log \sum_x (p_x)^2
\]
When is extractor guaranteed to exist?

Forget about \( \delta \) for the moment,

\[
\Delta(F(X, R)Y R; U\ell Y R) \leq \frac{1}{2} \sqrt{\delta \times 2^{\ell - \hat{H}_2(X|Y)} - 2^{\ell - \hat{H}_2(X|Y)}}
\]

Set this to \( \varepsilon \)

\[
\text{If } \ell \leq \hat{H}_2(X|Y) + 2 - 2 \log \frac{1}{\varepsilon} \text{ then UHF gives Stat.dist } \leq \varepsilon
\]

Quality of the source

Penalty for demanding \( \varepsilon \)-uniformity

Lots of entropy wasted!
q-wise independent hashing

- Very recent result [Dodis et al. 2014].
- Limited use compared to UHF
  - MACs, signatures, keyed hashes

Definition similar to UHF:

**Definition 3.15** A q-wise independent family of hash functions from $\mathcal{X}$ to $\mathcal{Y}$ is a set $\{h_s\}_{s \in S}$ of functions $h_s : \mathcal{X} \rightarrow \mathcal{Y}$ with the following property,

$$\forall_{\text{distinct } x_1, \ldots, x_q \in \mathcal{X}} \forall_{y_1, \ldots, y_q \in \mathcal{Y}} \Pr[h_S(x_1) = y_1 \land \cdots \land h_S(x_q) = y_q] = |\mathcal{Y}|^{-q}. \quad (3.20)$$
Start with an algorithm that has "security $\delta$" when used with a perfect key.

Compress to size $\ell \leq H_\infty(X) - 4 - \log \log \frac{1}{\varepsilon}$

using $q = 6 + \lceil \log \frac{1}{\varepsilon} \rceil$.

Then the security of the algorithm goes from $\delta$ to $\delta' = 2\delta + \varepsilon$. 