Random Numbers

Elisabeth Oswald

OUTLINE

What are random numbers?

How to create random numbers

How to test if numbers are random

Over to you

Overview

What are random numbers?

How to create random numbers

How to test if numbers are random

Over to you

PROPERTIES OF RANDOM NUMBERS

The concept of randomness is familiar to many of us, and this can imply misconceptions about its actual mathematical meaning.

Firstly, a single number cannot be random as such. Only a process by which we repeatedly select numbers can have the property to select numbers at random.

Secondly, random is not the same as arbitrary. Consider for example the instruction to "press any key". This instruction means that any key on the keyboard can be selected. If this instruction is issued repeatedly, then a user could always press the key with label "a". In contrast the instruction "press a random key" would imply that upon repetition of this instruction, the user would need to select a key at random and therefore always pressing the same key would not be o.k.

Properties of random numbers

From the previous slide we can deduce that the concept of randomness implies we are talking about a process that involves a set of elements, and that upon repetition of the process it is impossible to know how the outcome of the next invocation will be given knowledge about all previous invocations.

Thus given a set, and a distribution over the set, drawing an element from this set at random means that:

- over many times drawing an element the distribution of the drawn elements is indistinguishable from the distribution of the underlying set (the elements appear "unbiased")
- we probability for drawing any element is independent of the previously drawn elements (the elements are "unpredictable")

PROPERTIES OF RANDOM NUMBERS: EXAMPLE

We choose our set of numbers as bits $\{0, 1\}$, with probability $p(0) = p(1) = 0.5$.

Then in an experiment where we draw bits from random from this set we expect that (in a sufficiently long run), all sampled bits have again the distribution $p(0) = p(1) = 0.5$. We would also expect that any sequence of bits such as $00, 01, 10, 11$ occurs with equal probability (i.e. 0.25 in this case).

If the individually sampled bits (over a long enough sequence) are equiprobable, and any sequence of them is equiprobable, the bits would be unbiased.

If we can also show that sequences are independent, then the sampled bits would also be unpredictable.

PROPERTIES OF RANDOM NUMBERS FOR CRYPTOGRAPHIC PURPOSES

If we use random numbers to generate keys, in which case we really need both unbiasedness and unpredictability.

If we use random numbers as IVs, then we also need both unbiasedness and unpredictability.

For nonces we need unbiasedness, but not necessarily unpredictability.

Overview

What are random numbers?

How to create random numbers

How to test if numbers are random

Over to you

Adversarial assumptions

> Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin. — John von Neumann

The criterion for "unpredictability" requires us to think a little bit more:

- if an adversary would have unlimited computational power (i.e. the same assumptions that we postulated for "unconditional security"), then of course von Neumann is correct and there is no way that we could produce random numbers algorithmically,
- if we don't assume that an adversary has unlimited computational power then perhaps we could formulate a notional of "computational indistinguishability" where we'd hope that an adversary couldn't distinguish a given sequence of numbers from a sequence of truly random numbers (with their computational power)
- where would we get truly unpredictable numbers from?

True randomness vs Pseudo randomness

Pseudo RNG (PRNG): in an asymptotic setting (i.e. considering suitably long sequences), a function is a PRNG if its output is computationally indistinguishable from true randomness.

True RNG (TRNG): is a generator that is based on a process that is inherently probabilistic and sufficiently complex so it unpredictable.

There are a range of sources for true randomness based on physical processes, and it is imperative that they are checked for their distribution (in cryptography we often need uniform distributions). It is also important to ensure that they cannot be manipulated.

True RNGs

Depending on the application, a range of sources of natural randomness are available:

- elapsed time between emission of particles during radioactive decay;
- thermal noise from a semiconductor diode or resistor;
- the frequency instability of a free running oscillator;
- the amount a metal insulator semiconductor capacitor is charged during a fixed period of time;
- sound from a microphone or video input from a camera; signals from an antenna.

Some sources require specialist equipment and one cannot embed them on commodity devices; some sources can be manipulated by adversaries and therefore require special consideration.

Typical high-end products for security will include a hardware TRNG (often based on free running oscillators).

Publicity Friendly True RNGs

"To collect this data, Cloudflare has arranged about 100 lava lamps on one of the walls in the lobby of the Cloudflare headquarters and mounted a camera pointing at the lamps. The camera takes photos of the lamps at regular intervals and sends the images to Cloudflare servers. All digital images are really stored by computers as a series of numbers, with each pixel having its own numerical value, and so each image becomes a string of totally random numbers that the Cloudflare servers can then use as a starting point for creating secure encryption keys."

Figure: Cloudflare Lava lamps

`https://www.cloudflare.com/en-gb/learning/ssl/lava-lamp-encryption/`

Money Making True RNGs

" RANDOM.ORG offers true random numbers to anyone on the Internet. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs. People use RANDOM.ORG for holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music. "

Figure: Random.org, By Source, Fair use, https://en.wikipedia.org/w/index.php?curid=24802848

`https://www.random.org`

Suspence inducing True RNGs

"A random number generator assembly for mixing and randomly selecting balls indicating events or numbers is described which includes a chamber for mixing the numbers and a storage device communicating therewith for storing the numbers which have been generated. The mixing chamber is constructed from transparent plastic forming an octagonal drum. Motion of the balls within the mixing chamber is provided by a blower having sharply bent blades rotated with the convex side in the direction of travel to displace air within the mixing chamber. Within the mixing chamber, a multi-planar ramp made of expanded metal serves to increase the mixing effect of the balls providing more complete and rapid randomization of the balls. "

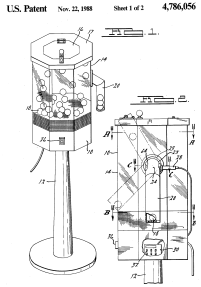

Figure: Lottery RNG Design by Dunnigan, https://patents.google.com/patent/US4786056

Delicate RNGs

Not all systems have dedicated hardware TRNGs available and therefore people have suggested all sorts of workarounds (taken from HAC chapter 5):

- the system clock;
- elapsed time between keystrokes or mouse movement;
- content of input/output buffers;
- user input; and
- operating system values such as system load and network statistics

I call these sources "delicate" because they can be potentially quite easily manipulated by adversaries (who may run a process on the target machine). Also some of these processes are not really random as in unpredictable (think of the system clock: it changes rather predictably).

OSs typically harvest from "all" of these sources to create a randomness pool from which applications can harvest some random "bits". An analysis of the Linux RNG strategy can be found here `https://web.archive.org/web/20081003041432/http://www.pinkas.net/PAPERS/gpr06.pdf`.

Practice: combining TRNGs and PRNGs

Any TRNG "harvests" entropy over time; thus only once enough entropy has been generated, one can actually ask the TRNG for a random number (i.e. bit sequence of a specified length).

But many crypto applications require a "lot of" randomness: e.g. for fresh session keys (authenticated encryption) plus perhaps an IV; and then the same every time some re-keying happens.

Thus in practice we use a TRNG to generate the seed for a PRNG: the latter then produces pseudo-randomness quickly.

PRNG Design Strategies

A simple way of creating pseudo-random numbers is to use some form of linear congruence (i.e. a simple equation over a finite group/ring/field). This often leads to something called LFSR (linear feedback shift register).

The problem with this approach is that the resulting numbers are not unpredictable (not even for a computationally limited adversary); although for suitably chosen LFSRs the resulting numbers are unbiased (thus they are useful for testing and experimentation).

Thus for crypto we need too look for more involved design strategies, and perhaps unsurprisingly, we can re-use some crypto primitives to build cryptographically secure PRNGs.

NIST RECOMMENDED PRNG CONSTRUCTIONS: HMAC-DRBG

HMAC: Mac based on Hash function, DRBG: Deterministic random bit generator

A string of random bits then gives a random number.

HMAC-DRBG is HMAC in OFB mode. Seed is taken from some TRNG. Key is either fixed between users (not practicable) or session key is generated/exchanged alongside.

NIST RECOMMENDED PRNG CONSTRUCTIONS: HASH-DRBG

HASH-DRBG uses a hash function in counter mode. Seed is taken from some TRNG. The counter is increased and some stuff (including the hash from the previous call) is added to it before it can be used as seed.

NIST RECOMMENDED PRNG CONSTRUCTIONS: BASED ON A BLOCK CIPHER

Similar to the constructions based on a hash function, it is also possible to use a block cipher in a mode of operation to create randomness.

NIST recommends to use either AES or Triple-DES (and cautions w.r.t. the latter because of its too small block size).

The Dual_EC_DRBG Scandal

NIST defined more DRBGs, one being based on the DL problem over elliptic curves.

Elliptic curve crypto was meant to replace RSA based and conventional DL based crypto because the DL problem over an elliptic curve can be substantially harder and therefore one can use shorter keys to achieve the same level of difficulty than when using RSA/DL.

But this particular DRGB was not faster than other DRGBs and the NSA "bribed" RSA Labs (back then a large influential company in the cybersecurity business) to make this DRGB their "default" choice.

This seemed iffy because already in 2006 researchers pointed out that the random numbers from this construction were actually slightly biased:
`https://eprint.iacr.org/2006/190`.

The Dual_EC_DRBG Scandal

But then some other researchers demonstrated a possible attack that could imply a deliberate backdoor.

The construction (bare bones) can be sketched like this. Let P and Q be points on an elliptic curve over some suitable finite field; s_i be the current state and $x()$ be the function that extracts the x coordinate from an EC point, and x_{16} the function that extracts all but 16 bits from the EC point. The dual EC construction then proceeds as follows.

$$s_{i+1} = x(s_i P)$$
$$o_i = x_{16}(s_i Q)$$

The Dual_EC_DRBG Scandal

Now let's consider this as an attack: the adversary is assumed to have selected the points P and Q carefully such that $dQ = P$. They keep d secret but push P and Q to be the public parameters in an implementation.

Now somebody uses this construction with an initial seed s_0 that is unknown to the adversary.

But the adversary can see the output o_i at some point. This implies they know all but 16 bits of the x coordinate of $s_i Q$. Therefore they can enumerate all possibilities for this x coordinate and test which of them lead to actual EC curve points (i.e. which ones do have a valid y coordinate). They will get a set of candidate EC points. For each of these candidates they can compute the next state because $ds_i Q = s_i P$.

When they then get information about the next state via o_{i+1} they can in fact confirm what was the correct of the candidates and with that they know the current state of the RNG. As a result they can predict any following states until the RNG gets reseeded.

The Dual_EC_DRBG Scandal

So why does this suggest a backdoor?

In the NIST standard some P and Q are specified. But nobody knows where these points came from. And because the DL problem on an EC is hard, we cannot check if there is indeed a d such that $dQ = P$.

NIST also acknowledged the contributions made by the NSA to this standard.

Therefore there is the widely shared opinion that the construction was deliberate and part of the wider effort of the NSA to undermine cryptographic standards.

The codename for NSA operation to undermine crypto standards is “Bullrun”:
`https://en.wikipedia.org/wiki/Bullrun_(decryption_program)`
(thanks to Edward Snowden).
For more info see `http://dualec.org/`.

Overview

What are random numbers?

How to create random numbers

How to test if numbers are random

Over to you

Standard Tests

There are a range of standards for testing sequences of numbers w.r.t. their properties. We look at a few of them:

- Frequency test (single bit test)
- Serial test (two bit test)
- Poker test (extension of the single bit test to multiple bits)
- Runs test

They are based on testing whether a specific property is distributed “as expected” via a “goodness of fit” test.

FREQUENCY TEST

The purpose of this test is to determine whether the number of 0's and 1's in a given sequence are approximately the same. Let's denote the number of 0's by n_0 and the number of 1's by n_1. The total number of digits is then $n = n_0 + n_1$.

The following test statistic

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

is small if n_0 and n_1 are close, and it is large otherwise.

One can show that the X_1 follows a χ^2 distribution with 1 degree of freedom if $n \geq 10$.

SERIAL TEST

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of a given sequence are approximately the same. Let's denote the number of 0's by n_0, the number of 1's by n_1, the number of 00's by $n_0 0$ etc. . The total number of digits is again n.

The following test statistic

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_1^2 + n_1^2) + 1$$

is used. Note that $\sum_{i,j} n_{ij} = n - 1$ and $n_0 + n_1 = n$. Thus if all n_{ij} are about equally likely and both n_0 and n_1 are equally likely the test statistic is supposed to be very small.

One can show that the X_2 follows a χ^2 distribution with 2 degrees of freedom if $n \geq 21$.

Poker test and Runs test

The Poker test looks at the frequencies of longer sequences by chopping up the given sequence in k parts (non overlapping). The idea is that also for longer sequences, they should occur with a similar frequency.

The Runs test looks at runs of consequtive 0's and 1's. The distribution of such runs follows Golomb's randomness postulates and one can test for that.

Rather than getting bogged down in too many details we look at how such testing concretely works.

HYPOTHESIS TESTS

Hypothesis tests are a statistical tool to test some assumptions, e.g. the number of 0's and 1's are statistically close in this sequence.

Therefore we formulate a "null hypothesis" and an alternative hypothesis:
H_0 : the number of 0's and 1's are very close vs.
H_1 : there is a significant difference between the number of 0's and the number of 1's.

If two random variables are statistically close their distributions must be very similar.

For discrete variables we can use the following test statistic to determine if the distribution of an observed variable O is close to an expected distribution E :

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

The Frequency Test is a Goodness of Fit Test

The Frequency Test is a Goodness of Fit Test, Example

But what do we do with the resulting test statistic?

We know now how to "quantify" the difference between an observed and an expected distribution, but when does the actual numerical value indicate that the two distributions are actually different?

For this we need to know what is the probability that such a difference occurs. Intuitively, small differences should occur more often than a really large difference (if indeed the two distributions are similar).

With the test statistic we can compute such a probability by plugging it into the distribution itself.

Goodness of Fit Test, Table

For the χ^2 test statistic, the following table shows some probabilities for some degrees of freedom (typically this is replaced by a function call in a suitable statistics package).

	α					
v	0.100	0.050	0.025	0.010	0.005	0.001
1	2.7055	3.8415	5.0239	6.6349	7.8794	10.8276
2	4.6052	5.9915	7.3778	9.2103	10.5966	13.8155
3	6.2514	7.8147	9.3484	11.3449	12.8382	16.2662
4	7.7794	9.4877	11.1433	13.2767	14.8603	18.4668
5	9.2364	11.0705	12.8325	15.0863	16.7496	20.5150
6	10.6446	12.5916	14.4494	16.8119	18.5476	22.4577
7	12.0170	14.0671	16.0128	18.4753	20.2777	24.3219
8	13.3616	15.5073	17.5345	20.0902	21.9550	26.1245
9	14.6837	16.9190	19.0228	21.6660	23.5894	27.8772

Figure: Part of the Chi Square distribution table

Goodness of Fit Test, Table

This table has to be read as follows. The values for v represent the degrees of freedom that a test has. The degrees of freedom correspond to the number of independent variables. The α values give us the confidence value (a probability). E.g. if we want to ensure with 0.95 probability that two distributions are statistically very close then their X_1 value must not exceed 3.8415.

	α					
v	0.100	0.050	0.025	0.010	0.005	0.001
1	2.7055	3.8415	5.0239	6.6349	7.8794	10.8276

Figure: Part of the Chi Square distribution table

The Frequency Test is a Goodness of Fit Test, Example

Consider the following sequence:

$$1110001100010001010011101111001001001001$$
$$1110001100010001010011101111001001001001$$
$$1110001100010001010011101111001001001001$$
$$1110001100010001010011101111001001001001$$

This sequence is obviously not random. But it would pass the frequency test with flying colours:

$n_0 =$

$n_1 =$

$X_1 =$

Warning

Be careful to NEVER confuse statistical evidence with a mathematical proof: just because some experiment passes some statistical test, this is not proof of correctness.

In particular in statistics we would actually say that given n observations we did not find enough evidence to refute the null hypothesis at a certain confidence level.

This is even more cautious, but it really makes sense: a mathematical proof is always true, and this can only be guaranteed by some form of logical deduction or argument, but not by observing experimental outcomes. The only thing we can do with practical experiments is to disprove something: by collecting enough evidence to produce a counterexample.

Overview

What are random numbers?

How to create random numbers

How to test if numbers are random

Over to you

Complementary Reading and Watching

This course consists not just of material that I produced, but I also want you to benefit from a range of existing other materials from some fantastic researchers out there. Unfortunately in the case of random number generation, there is not very much good material available.

- Consider how to generate cryptographically secure random numbers in different languages. Perhaps visit `https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-lan c-csprng`.

The intended learning goals are that you know about the requirements for cryptographic randomness and how to generate such randomness.