Lecture 11: From OWFs to PRGs

Chris Brzuska, Christoph Egger

March 2024

1 Overview over techniques

Today, we are going to see how to construct pseudorandom generators from one-way functions. This overview is experimentally written, and we welcome feedback on the writing. We try to hint at core technical issues early on, before going into all the details of the constructions, all the parameters etc.. We personally dislike handling parameters and try to highlight what we find interesting and important. It might be that this first section is more readable after reading the rest of the lecture notes. In any case, let us know what you think (Exercise 1 on Exercise Sheet 10).

Pseudo-entropy. We'll start with the observation that the function

$$G(x) := f(x)||x$$

is a so-called *next-bit pseudo-entropy generator* if f is a one-way function. This means that given some prefix of the output of $G(x)$, it will be "somewhat hard" to guess the next bit. This makes sense, because if from every prefix of $G(x)$ the next bit is easy to determine, then given $f(x)$, we could just recover x bit-by-bit (Get x_1 from $f(x)$, get x_2 from $f(x)||x_1$, then x_3 from $f(x)||x_1||x_2$ and so on), so there need to be *some* bits which are hard to guess.

How many bits of x need to be hard to recover?

Well, if it's only logarithmically many, one could guess them with probability $2^{-\log n} = \frac{1}{n}$ which would be too high probability, since it would allow to invert the one-way function also with probability $\frac{1}{n}$ (which is non-negligible).

So, there need to be slightly more than logarithmically many bits of x which are hard to predict from $f(x)$ (and some prefix of x). To speak about the hardness of prediction or the *entropy* of the next bit, it will be useful to actually use formal notions of entropy. In Section 2, we revisit entropy, discuss the differences between *min-entropy* and its average-case version, the classic (but somehow more cumbersome) *Shannon entropy*. Moreover, we also define *next-bit entropy*, which will be a (computational) variant of Shannon entropy. Finally, we recall (a special case of) the leftover hash-lemma which gives us an extractor which takes as input (a public random input and) a random variable Z with min-entropy k and allows us to extract $k - \mathsf{superlog}(n)$ many bits from Z which are almost uniformly distributed, i.e., their statistical distance from the uniform distribution is $2^{-\mathsf{superlog}(n)}$ which is negligible in n.

Chernoff. Unfortunately, a one-way function f only ensures that G has *Shannon* next-bit entropy, but not min-entropy. This is annoying, because we want to build a PRG, so we want to use an extractor on something to get some nearly uniform bits. But extractors need min-entropy. Here, our friend Chernoff comes to help again (in a new variant), and we review the Chernoff bound which we need in Section 3. Exercise Sheet 10 also has a nice fill-in exercise for proving one variant of the Chernoff bound so that these Chernoff bounds become a little less obscure[1]. Maybe, in a future course, we also prove this more general variant, if we find a nice and concise proof of it.

We will see that using a general Chernoff bound (for independent random variables which are not Booleans), we can show that if we sample a couple of random variables $Z = (X_1, .., X_t)$ and look at all of their outputs together, then their min-entropy is quite similar to their "expected" uncertainty, a.k.a. Shannon entropy. Concretely, Chernoff bound helps us to show that if we have sufficiently many independent random variables, then their sum is close to their expectation. However, now, the random variables that we consider and sum of are the *entropies* of the X_i, i.e., we consider the expectation of

$$Z := \sum_i -\log p_i(X_i),$$

where $p_i(z^i) := \Pr\big[X_i = z^i\big]$. The expectation $\mathbb{E}(Z)$ is equal to the sum of the Shannon entropies $H(X_i)$ and we find that with overwhelming probability, $\sum_i -\log p_i(X_i)$ is close to $\sum_i H(X_i)$.

Hybrid arguments. The proof uses two hybrid arguments. One of them is very similar to the one in Lecture 7 in that it iterates a PRG to get more and more random bits. However, this time, instead of iterating a PRG with 1 bit stretch, we iterate a so-called *Z-seeded PRG G*. G gets as input the output of a random variable Z and outputs a pair (Z', y) such that they are computationally indistinguishable from (Z, z), where z is a string drawn uniformly at random from $\{0,1\}^{|y|}$. In other words, the *seed* of G is *not* distributed uniformly at random, only the additional output y is distributed uniformly at random. In Section 4, we see[2] that we can use the exact same argument as in Lecture 7 to show that iterating this Z-seeded PRG yields a proper PRG.

The other hybrid argument is over the bits of $G_{nb}(x) = f(x)||x$. The reason is that we want to benefit from *each* bit of $G_{nb}(x) = f(x)||x$, but next-bit pseudo-entropy only states "Cut the last couple of bits of $G_{nb}(x)$ and then look at the entropy of the next bit", i.e., it is an argument only for a single bit. Thus, we use a hybrid argument, where step-by-step, we make the bits of $G_{nb}(x)$ "disappear" from the distribution, starting with the last bit, so that we can use entropy arguments where only previous bits are known.

[1] Spoiler: The proof of the Chernoff bound for n independent binary random variables X_i with $\Pr[X_i = 1] = p$ is simply by exponentiating both sides of the inequality $\sum X_i > pn + \epsilon n$ and then applying the Markov bound which sais that for each positive $a > 0$ and any random variable Z, $\Pr[Z \geq a] \leq \frac{\mathbb{E}[Z]}{a}$.

[2] Well, we'll actually omit the proof, but hopefully, you can get convinced yourself...

Averaging. For the last approach, consider the case that the last bit of $G_{nb}(x)$ has no next-bit pseudoentropy whatsoever, e.g., it might be that it is a copy of the first bit of $G_{nb}(x)$. Then, we cannot extract randomness from that bit. Therefore, we will average over all positions of $G_{nb}(x)$ in the proof. See Section 6 for a description of the construction of the Z-seeded PRG.

2 Notions of Entropy

2.1 Min-Entropy

Entropy measures how "random" a distribution is, and it measures entropy in *bits*. I.e., all entropy measures are defined such that the uniform distribution over n bits will have entropy n. One of the most important notions of entropy in cryptography is *min-entropy*. It measures the *worst-case* entropy, i.e., looks at the most likely element in a distribution.

Definition 1 (Min-Entropy)**.** Let $\mathcal{D}$ be a distribution. Then, its min-entropy $H_\infty(\mathcal{D})$ is defined as

$$H_\infty(\mathcal{D}) := -\log_2 \max_x(\Pr[\mathcal{D} = x]).$$

Example. Consider $\mathcal{D}_{\text{example}}$ which returns

$$\begin{aligned} 0^n \text{ with probability } \frac{1}{2} \\ \text{a uniformly random string from } \{0,1\}^n \text{ with probability } \frac{1}{2}. \end{aligned}$$

We can compute the min-entropy of $H_\infty(\mathcal{D}_{\text{example}})$ by looking at its most likely element, which is 0^n and occurs with probability $\frac{1}{2}$ (well, actually, even with slightly higher probability $\frac{1}{2} + 2^{-n}$, but let us ignore this for sake of simplicity). Then, $\log_2(\frac{1}{2})$ is equal to -1 and $-(-1) = 1$, so $\mathcal{D}_{\text{example}}$ has 1 bit of min-entropy, i.e., $H_\infty(\mathcal{D}_{\text{example}}) = 1$.

2.2 Shannon Entropy

Looking at $\mathcal{D}_{\text{example}}$, we might think that min-entropy is a rather wasteful and, in some way, inaccurate way of measuring "how much randomness" is contained in a distribution, since it just ignores that there is a lot of randomness in the distribution $\mathcal{D}_{\text{example}}$. Maybe, it would be more accurate to consider the *average* entropy rather than the min-entropy. Shannon formalized this notion of average entropy and it is named after him.

Definition 2 (Shannon Entropy)**.** Let $\mathcal{D}$ be a distribution. Then, its Shannon entropy $H(\mathcal{D})$ is defined as

$$H(\mathcal{D}) := -\sum_x \Pr[\mathcal{D} = x] \cdot \log_2(\Pr[\mathcal{D} = x]).$$

Note that now, in the formula for $H(\mathcal{D})$, we replaced $\max_x$ by $\sum_x \Pr[\mathcal{D} = x]$, i.e., we now average over all possible values x and weigh by their probability. How much Shannon entropy does our example $\mathcal{D}_{\text{example}}$ have? Since the all-zero string (at least intuitively) does not contribute much entropy, let's look at the other strings. Their probability is $\frac{1}{2} \cdot 2^{-n}$. Now, for any of such string x, we have

$$\log_2(\Pr[\mathcal{D} = x]) = \log_2(\tfrac{1}{2} \cdot 2^{-n}) = -(n+1).$$

For these strings, $\sum_x \Pr[\mathcal{D} = x]$ sums up to (almost) $\frac{1}{2}$ (We would again need to remove the all-zero string and would only get $\frac{1}{2} - 2^{-n}$, but let's ignore this.), and so, we obtain that the Shannon entropy of $\mathcal{D}_{\text{example}}$ is $H(\mathcal{D}_{\text{example}}) \approx \frac{n+1}{2}$.

2.3 Next-Bit Pseudo-entropy

A next-bit pseudo-entropy generator G_{nb} is a function where it is hard to predict *some* of the bits, given the previous bits of the output of the function, i.e., there are some random variables Y^i such that they are indistinguishable from the actual next bit of G_{nb}

$$G_{nb}(x)[1..i] \overset{\text{comp}}{\approx} G_{nb}(x)[1..i-1]||Y^i$$

and such that the Y^i have *real* entropy, i.e., the sum of the entropies of Y^i is higher than the entropy of x. In particular, we will ask that if x is a uniformly random bitstring of length n, then the sum of the entropies of the Y^i should be $n + \log_2(n)$, that is, $\log_2(n)$ bits greater than the actual entropy.

Definition 3 (Pseudo-entropy Generator)**.** A polynomial-time computable, deterministic function $G_{nb} : \{0,1\}^* \to \{0,1\}^*$, with $\forall x \in \{0,1\}^* \, absG_{nb}(x) = 2|x|$ has $n + \log_2 n$ bits *next-bit pseudo-entropy* if for all $n \in \mathbb{N}$, there are random variables $Y^1, .., Y^{2n}$ with $Y^i \in \{0,1\}$ and jointly distributed with x such that the following holds:

Entropy $\sum_{i=1}^{2n} \mathbb{E}_{x \leftarrow\$ \{0,1\}^n}[H(Y^i | G_{nb}(x)[1..i-1])] = n + \log_2 n$

Indistinguishability For all pairs of PPT $\mathcal{A}_1, \mathcal{A}_2$, the following advantage is negligible in n:

$$\begin{aligned} &|\Pr_{x \leftarrow\$ \{0,1\}^n, i \leftarrow\$ \mathcal{A}_1(1^n)}\left[\mathcal{A}_2^{\mathsf{O}(\cdot)}(1^n, G_{nb}(x)[1..i])\right] \\ &- \Pr_{x \leftarrow\$ \{0,1\}^n, i \leftarrow\$ \mathcal{A}_1(1^n), y \leftarrow\$ Y^i}\left[\mathcal{A}_2^{\mathsf{O}(\cdot)}(1^n, G_{nb}(x)[1..i-1]||y)\right]|, \end{aligned}$$

where $\mathsf{O}(j)$ can be called repeatedly, samples $x' \leftarrow\$ \{0,1\}^n$ and $y' \leftarrow\$ Y^j$ and returns $G_{nb}(x')[1..j-1]||y'$. $G_{nb}(x)[1..i]$ denotes the first i bits of $G_{nb}(x)$.

First, recall that we can assume without loss of generality that a one-way function is length-preserving (by appending zeroes or ignoring some input bits), see Goldreich, *Foundations of Cryptography I* for a complete proof.

Lemma 1 (Length-preserving OWFs (Goldreich))**.** If OWFs exist, then length-preserving OWFs exist.

Lemma 2 (Vadhan-Zheng)**.** If f is a length-preserving one-way function, then

$$G_{nb}(x) := f(x)||x$$

has next-bit pseudo-entropy $n + \log_2 n$.

We will not prove this lemma (at least in this year's edition of the course), since we do not feel sufficiently comfortable with the notion of KL-divergence yet which is used in the proof of the lemma. However, we refer back to Section 1 for an explanation of why $n + \log_2 n$ is a plausible bound. Interestingly, Miikka Tiainen proved in his master thesis, that the same statement is true if f is a *distributional* one-way function.

Lemma 3 (Tiainen)**.** If f is a length-preserving distributional $(1 - \mathsf{negl}(n))$ one-way function, then

$$G_{nb}(x) := f(x)||x$$

has next-bit pseudo-entropy $n + \log_2 n$.

For the rest of this lecture, we will work based on a function G_{nb} which has $n + \log_2 n$ bits pseudo-entropy.

2.4 Randomness Extractors

Recall that a randomness extractor takes the entropy of a distribution and turns it into something uniformly random. Randomness extractors are rather picky and needy—they only work with the *min-entropy* of a distribution. Moreover, they also need a *public* uniformly random and independent salt. Compare Lecture 5 for a more extensive discussion of randomness extractors, their needs and limitations. We now recall the definition of statistical distance, then the definition of extractors and then the leftover hash-lemma (cf. Lecture 5).

Definition 4 (Statistical Distance)**.** For two random variables X and Y, we define the statistical distance between X and Y as

$$\mathsf{SD}(X, Y) := \tfrac{1}{2} \sum_{z \in \mathrm{Supp}(X) \cup \mathrm{Supp}(Y)} |\Pr[X = z] - \Pr[Y = z]|,$$

where the support $\mathrm{Supp}(X)$ denotes the set of values z where $\Pr[X = z] > 0$.

Definition 5 (Strong extractor)**.** A function $\mathsf{ext} : \{0,1\}^r \times \{0,1\}^\ell \to \{0,1\}^m$ is an (k, ϵ)-strong extractor if for all random variable X with $H_\infty(X) \geq k$, we have that

$$\mathsf{SD}((\mathsf{ext}(X, S), S), (U_m, S)) \leq \epsilon.$$

Here, the set R_m is a set of randomness which depends on m.

Alternative formulation of indistinguishability One might also define an extractor by requiring that all adversaries (even inefficient ones) have at most distinguishing advantage ϵ in distinguishing the following two experiments.

$\text{Real}_{\mathcal{D},\mathsf{ext}}$	$\text{Real}_{\mathcal{D},\mathsf{ext}}$
$x \leftarrow\!\!\$\ \mathcal{D}$	
$s \leftarrow\!\!\$\ \{0,1\}^\ell$	$s \leftarrow\!\!\$\ \{0,1\}^\ell$
$y \leftarrow \mathsf{ext}(s,x)$	$y \leftarrow\!\!\$\ \{0,1\}^m$
return (s,y)	**return** (s,y)

Lemma 4 (Leftover Hash Lemma, Proof in Lecture. 5)**.** Let t and n be natural numbers. Let $h : \{0,1\}^{tn} \times \{0,1\}^\ell \to \{0,1\}^m$ be a 2-universal hash-function. Then h is also a strong (k,ϵ) extractor as long as $m \leq k - 2 \cdot \log_2(\frac{1}{\epsilon})$.
In particular, for $m \leq k - \mathsf{superlog}(n)$, $\epsilon(n)$ is negligible in n.

We only added the text in pink to the lemma as compared to Lecture 5 and replace the input length by tn instead of n to match it to the scenario which we will use later in the proof today.

3 Chernoff Bound

Lemma 5 (Chernoff)**.** Let X_1,..,X_m be m independent random variables such that for all i, $0 \leq X_i \leq \ell$ and such that $\mathbb{E}[\sum_i X_i] = \mu$. Then, for all δ,

$$\Pr\left[\sum_{i=1}^{m} X_i \leq (1-\delta)\mu\right] \leq e^{-\frac{\delta^2\mu^2}{m\ell^2}} \tag{1}$$

In particular, for $\mu = t(n+\log_2(n))$, $\ell = 2\mathsf{log}(n)$, $m = tn$ and $\delta = \frac{\log_2(n) - \frac{\mathsf{superlog}(n)}{t}}{n+\log_2(n)}$, we have that

$$\begin{aligned}
(1-\delta)\mu &= (1-\delta)t(n+\log_2(n)) \\
&= \left(1 - \frac{\log_2(n) - \frac{\mathsf{superlog}(n)}{t}}{n+\log_2(n)}\right) t(n+\log_2(n)) \\
&= \left(\frac{n+\log_2(n) - \log_2(n) + \frac{\mathsf{superlog}(n)}{t}}{n+\log_2(n)}\right) t(n+\log_2(n)) \\
&= \left(\frac{n + \frac{\mathsf{superlog}(n)}{t}}{n+\log_2(n)}\right) t(n+\log_2(n)) \\
&= \left(n + \frac{\mathsf{superlog}(n)}{t}\right) t \\
&= tn + \mathsf{superlog}(n)
\end{aligned} \tag{2}$$

and

$$
\begin{aligned}
e^{-\frac{\delta^2\mu^2}{m\ell^2}} &= \mathsf{exp}\left(-\frac{1}{tn}\left(\frac{(\log_2(n)-\frac{\mathsf{superlog}(n)}{t})t(n+\log_2(n))}{(n+\log_2(n))\cdot 2\mathsf{log}(n)}\right)^2\right)\\
&= \mathsf{exp}\left(-\frac{1}{tn}\left(\frac{(\log_2(n)-\frac{\mathsf{superlog}(n)}{t})t}{2\mathsf{log}(n)}\right)^2\right)\\
&= \mathsf{exp}\left(-\frac{1}{tn}\left(\frac{(t\log_2(n)-\mathsf{superlog}(n))}{2\log_2 n}\right)^2\right)\\
&= \mathsf{exp}\left(-\frac{1}{2n^2\mathsf{superlog}(n)}\left(\frac{(2n\log_2(n)\cdot\mathsf{superlog}(n)-\mathsf{superlog}(n))}{2\log_2 n(n)}\right)^2\right) \text{ using } t=2n\cdot\mathsf{superlog}(n)\\
&\leq \mathsf{exp}\left(-\frac{1}{2n^2\mathsf{superlog}(n)}\left(\frac{n\log_2(n)\cdot\mathsf{superlog}(n)}{2\log_2 n}\right)^2\right)\\
&= \mathsf{exp}\left(-\frac{1}{8n^2\mathsf{superlog}(n)}\left(n\cdot\mathsf{superlog}(n)\right)\right)^2\\
&= \mathsf{exp}\left(-\frac{\mathsf{superlog}(n)}{8}\right) = \mathsf{negl}(n)
\end{aligned}
$$

4 Z-seeded PRG

Definition 6 (Z-seeded PRG)**.** Let $\mathcal{Z}(1^n)$ be an efficiently sampleable distribution. Let G be a polytime-computable deterministic function. G is a $\mathcal{Z}$-seeded pseudorandom generator if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible:

$$\left|\Pr_{z\leftarrow\$\mathcal{Z}(1^n)}[\mathcal{A}(1^n,G(z))]-\Pr_{z\leftarrow\$\mathcal{Z}(1^n),r\leftarrow\$\{0,1\}}[\mathcal{A}(1^n,z||r)]\right|$$

Lemma 6 (Z-seeded PRG)**.** Let $\mathcal{Z}$ be a poly-time sampleable distribution which gets as input a security parameter 1^λ and uses $p(\lambda)$ uniformly random bits to sample its output, where $p(\lambda)$ is a polynomial and also (w.l.o.g.) injective and such that given $p(\lambda)$, computing $\lambda := p^{-1}(p(\lambda))$ is easy.

If G is a Z-seeded PRG, then the following function $\mathcal{PRG}$ is a standard PRG.

$\mathcal{PRG}(r)$

$s \leftarrow$ empty string
$\lambda \leftarrow p^{-1}(|r|)$
$z \leftarrow \mathcal{Z}(1^\lambda; r)$
for $i = 1..|r|+1$
  $y \leftarrow G(z)$
  $z \leftarrow y[1..|r|]$
  $s \leftarrow s||y[|r|+1]$
return s

The proof of Lemma 6 follows exactly the same hybrid proof strategy which we also used for length-expansion in an earlier lecture. We thus omit the proof.

5 Main Theorem and Proof

Theorem 1. If one-way functions exist, then PRGs exist.

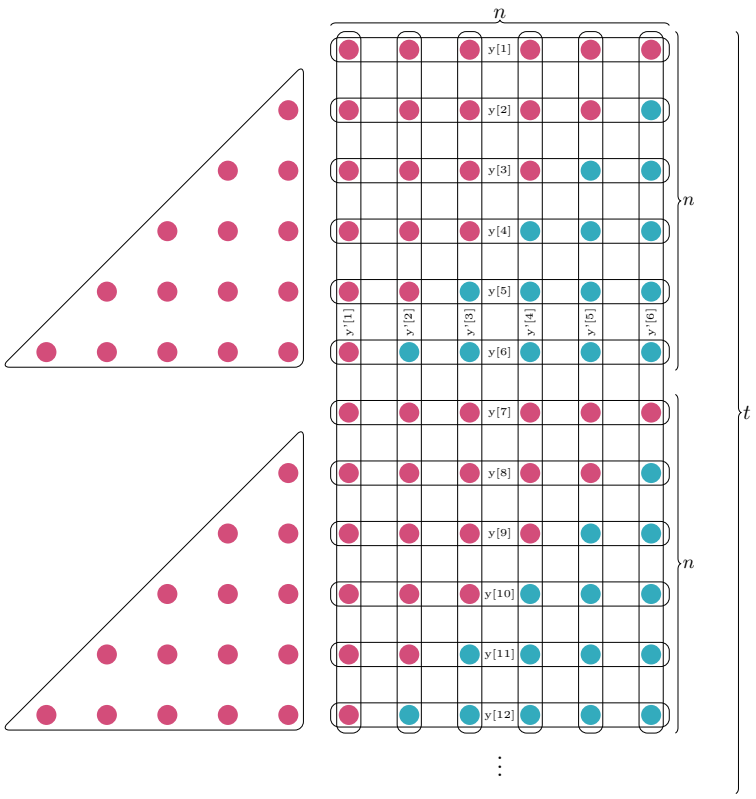
The proof consists of three main steps. Note that in the lecture video, we referred to these steps as Lemma 1, Lemma 2 and Lemma 3, but now here, the numbering is different. The first lemma we rely on is Lemma 2 which states that if there is a one-way function, then there is a poly-time computable function G_{nb} which is length-doubling and has $n+\log_2 n$ bits of next-bit pseudo-entropy. The second lemma we need is that from such a next-bit pseudo- entropy generator G_{nb}, we can construct a $\mathcal{Z}$-seeded PRG. The proof of this statement is rather complex and we thus call it a theorem, provide a construction in Section 6 and prove its security in Section **??** (not yet in the lecture notes, sorry!).

Theorem 2. If G_{nb} is a next-bit pseudo-entropy generator, then the construction G provided in Section 6 is a $\mathcal{Z}$-seeded PRG.

Finally, once we have a $\mathcal{Z}$-seeded PRG, we can use Lemma 6 to turn the $\mathcal{Z}$-seeded PRG into a standard PRG. Thus, all which remains to show is Theorem 1

6 Construction

In this section, we provide the construction of the $\mathcal{Z}$-seeded PRG G. The picture below illustrates the main idea. See the lecture video for motivation. We here provide the pseudo-code for formality.



We first describe the distribution $\mathcal{Z}$.

$\mathcal{Z}(1^n)$ ..

for $i = 1..2n$ ..

$x^i \leftarrow\$ \{0,1\}^n$

$w^i \leftarrow\$ \{0,1\}^n$

$h \leftarrow\$ \mathcal{H}$

// hash-function

// Construction of square

for $i = 1..2n$

for $\tau = 1..t$

for $i = 1..2n$

if $\tau > i$: $a_(\tau, i)$ $a_(\tau, i) G_{nb}(x)$