NYU COMPUTER SCIENCE FALL 2026

Introduction to Cryptography

CSCI-GA.3210-001 Instructor: Nir Bitansky

Lecture 4: Pseudorandomness and One-Way Functions

1 Previously on Introduction to Crypto

Starting from an encryption scheme for messages of length $\ell(n) = n + 1$ with n-bit keys, we constructed a scheme for messages of any polynomial length $\ell'(n) = n^{O(1)}$, still with n-bit keys. This fits within our goal of reducing cryptography, and in particular encryption, to a few basic primitives that we could hopefully base on a variety of assumptions. So far, we have reduced secret-key encryption for arbitrarily long messages to "one-extra-bit encryption" (OEB encryption), i.e., encryption for messages of length $n+1$ using n-bit keys. *But where would OEB encryption come from?*

We now wish to go further down the ladder of reductions and find "the right primitives": primitives that suffice for encryption and can be constructed under various assumptions. A key concept in this endeavor is *pseudorandomness*.

2 Pseudorandomness

Our basic goal here is to take a few random bits and use them to generate a longer string that looks random. That is, we want a deterministic procedure G that takes n random bits and maps them to $n + \ell(n)$ bits, for some $\ell(n) > 0$. It's not hard to see that $G(s)$ cannot be uniformly distributed, or even statistically indistinguishable from uniform, even when s is chosen uniformly at random. As was the case for encryption, all that we ask is that $G(s)$ fools computationally bounded algorithms into thinking that it's random.

Definition 2.1 (Pseudorandom Generator (PRG))**.** *Let U_k denote the uniform distribution on k-bit strings. A pseudorandom generator with stretch $\ell(\cdot) > 0$ is a deterministic polynomial-time algorithm G that maps n bits to $n + \ell(n)$ bits and whose output is pseudorandom:*

$$\{G(U_n)\}_n \approx_c \left\{U_{n+\ell(n)}\right\}_n \ .$$

The random input to the generator is called a **seed**.

Remark: Other Flavors of Pseudorandomness. In a wider context, PRGs as defined above are called cryptographic pseudorandom generators. They are quite ambitious in the sense that the same PRG should fool a very large class of algorithms — all efficient ones. As we'll see, their existence requires cryptographic assumptions. Pseudorandomness is widely studied beyond cryptography, mostly in the context of derandomization. In that context, we may gain a lot — in terms of assumptions or seed length — if we're only interested in fooling restricted classes of algorithms, such as algorithms with a fixed a priori running-time bound, space-bounded algorithms, or linear functions. In this course, we won't focus on non-cryptographic PRGs, but we should keep this distinction in mind.

In crypto, the purpose of PRGs is not to derandomize algorithms (in the sense of making them completely deterministic), but rather to allow cryptographic keys to be short, or more generally to *recycle randomness*. The first thing we would like to understand is their relation to OEB encryption.

Theorem 2.2. *PRGs with one-bit stretch ($\ell = 1$) exist if and only if OEB encryption exists.*

Corollary 2.3 (Of the theorem proved in the previous lecture)**.** *If PRGs with one-bit stretch exist, then there exists encryption for messages of any polynomial length $\ell(n) = n^{O(1)}$ with keys of length n.*

For now, we will prove only the first direction: PRGs imply OEB encryption. We will return to the converse later and prove a weaker statement.

PRGs with one-bit stretch imply OEB. Sample a uniform seed $s \leftarrow \{0,1\}^n$ and set the secret key to $sk := s$. Encryption of a message $m \in \{0,1\}^{n+1}$ is given by:

$$E_{sk=s}(m) = m \oplus G(s) \ .$$

Decryption of a ciphertext ct is given by

$$D_{sk=s}(ct) = ct \oplus G(s) \ .$$

This construction follows the intuition that a pseudorandom string is as good as a random string; in particular, we can use it in place of a one-time pad. Let's prove that this is secure, using the ensemble notation we saw in the previous lecture:

$$\{E_{U_n}(m)\}_{\substack{n\in\mathbb{N} \\ m\in\{0,1\}^{n+1}}} = \{m \oplus G(U_n)\}_{\substack{n\in\mathbb{N} \\ m\in\{0,1\}^{n+1}}} \approx_c \{m \oplus U_{n+1}\}_{\substack{n\in\mathbb{N} \\ m\in\{0,1\}^{n+1}}} \equiv \{U_{n+1}\}_{\substack{n\in\mathbb{N} \\ m\in\{0,1\}^{n+1}}} \ .$$

(Make sure you understand why the second and third ensembles are computationally indistinguishable.) □

Longer Stretch. Not surprisingly, given a PRG with stretch $\ell = 1$, we can get a PRG with any polynomial stretch $\ell'(n) = n^{O(1)}$. The construction and its proof are very similar to those used to extend encryption. The idea is illustrated below:

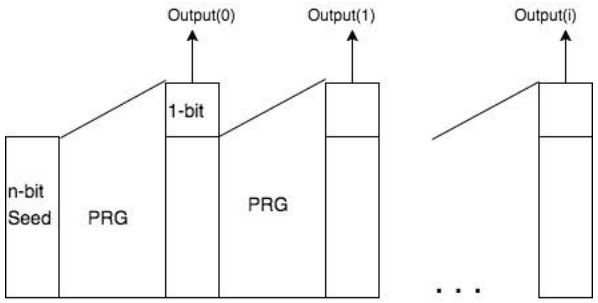
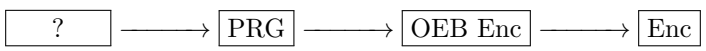


Figure 1: PRG-stretching illustration

Given a uniform seed $s_0 := s$, we apply the PRG to obtain $G(s_0) = \sigma_1 s_1$, where $|\sigma_1| = 1$ and $|s_1| = n$. We then apply the PRG to s_1 to obtain $G(s_1) = \sigma_2 s_2$, and continue for a total of $\ell'(n)$ iterations. We output $\sigma_1 \ldots \sigma_{\ell'} s_{\ell'}$, a string of length $n + \ell'(n)$. Security follows from a hybrid argument.

2.1 Where Do PRGs Come From?

Our web of reductions is starting to build:



Remember that as the source for this graph of primitives, we would like to have a primitive that we can base on a variety of hard problems. PRGs bring us closer to this, and already have direct (candidate) constructions:

1. **Practical Designs:** There exist many practical constructions for PRGs (see this list, for example). They are mainly designed to be *fast*, and typically we do not know how to reduce their security to a better assumption than "they are secure". Sometimes, we can show that they resist certain specific attacks. Sometimes they're broken. In the context of this course, this falls short of what we want.

2. **PRGs from Well-Studied Computational Assumptions:** There are quite a few computational assumptions asserting that a specific distribution ensemble is pseudorandom. Examples include the decisional Diffie–Hellman (DDH), Learning with Errors (LWE), and Learning Parity with Noise (LPN) assumptions, which we might see later in the course. These problems have been extensively studied by the computer science and mathematics communities.

3 One-Way Functions: A Minimal Cryptographic Primitive

We are going to go further down the ladder of reductions, with the aim of broadening the collection of problems on which we can base PRGs and, in turn, encryption.

Definition 3.1 (One-Way Function (OWF))**.** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if it is computable in polynomial time and, for every non-uniform PPT adversary $A = \{A_n\}_n$, there exists a negligible function $\mu(\cdot)$ such that for all $n \in \mathbb{N}$:*

$$\Pr\left[A_n(f(x)) \in f^{-1}(f(x)) \;\middle|\; x \leftarrow \{0,1\}^n\right] \leq \mu(n) \ .$$

That is, a one-way function is easy to compute in the forward direction, but its output on a uniformly random input is hard to invert.

Hard-on-Average NP Problems with Solved Instances — What OWFs Capture. Unlike the notions we've seen so far, OWFs may seem like odd creatures. What do they capture? The answer is that OWFs capture the minimal form of computational hardness necessary for crypto. Let's try to understand in what sense. We've already seen that if P = NP, we can break encryption with short keys. However, as already hinted, it seems that we need more:

- **Average-Case Hardness:** Usually, when we talk about the hardness of NP problems, we mean hardness *in the worst case*: every efficient algorithm fails on some worst-case instance. However, in cryptography we do not know ahead of time who our adversary will be. We therefore need a way to sample instances that *every efficient algorithm* fails to solve, except with negligible probability.

- **Solved Instances:** It's not enough that we can simply generate hard instances. The good guys must have some advantage over the bad guys — they should know solutions to the hard instances they generate. For example, in encryption, they should be able to decrypt. In other words, we want to sample hard-on-average instances together with their solutions.

Definition 3.2 (Hard-on-Average NP Problem with Solved Instances)**.** *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP relation. We say that R is hard on average with solved instances if there exists a PPT sampler S such that:*

- *S* **samples solved instances:** *for any $n \in \mathbb{N}$,*

$$\Pr\left[(x,w) \in R \mid (x,w) \leftarrow S(1^n)\right] = 1 \ .$$

- *S* **samples hard instances:** *for any (non-uniform) PPT $A = \{A_n\}_n$ there is a negligible $\mu(\cdot)$, such that for any $n \in \mathbb{N}$,*

$$\Pr\left[w' \leftarrow A_n(x) \text{ and } (x,w') \in R \mid (x,w) \leftarrow S(1^n)\right] \leq \mu(n) \ .$$

The following is a relatively easy exercise (make sure you know how to solve it).

Claim 3.3. *OWFs exist if and only if hard-on-average NP problems with solved instances do.*

Proof Sketch. Given an OWF f, define $S(1^n)$ to sample $x \leftarrow \{0,1\}^n$ and output $(f(x), x)$. Conversely, given a sampler S for a hard-on-average NP problem with solved instances, define $f(r)$ to be the first component of $S(1^n; r)$, where r is the sampler's random string. □

As we shall see, one-way functions will indeed be necessary for essentially any crypto task we'll encounter in this course. In particular:

Theorem 3.4. *Computationally secure secret-key encryption with short keys implies OWFs.*

Proof Sketch. Let (E, D) be a computationally secure encryption scheme for messages of length $2n$, with keys of length n, and assume E uses $\ell = \ell(n)$ bits of randomness. We define $f : \{0,1\}^n \times \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^*$ as follows:

$$f(sk, m, r) = (E_{sk}(m; r), m) \ .$$

This function is defined for inputs of length $3n + \ell$, and we'll show it is one-way for such inputs. This is enough, as we can extend the function to all input lengths so that one-wayness also extends (you'll do this in your homework).

The intuition is that if an adversary could invert this function, it would still succeed with nearly the same probability if we replaced the encryption of the random message m with an encryption of the fixed message 0^{2n}. However, a ciphertext generated independently of m is unlikely to decrypt to m under any key, simply because there are many more messages than keys.

Formally, given an adversary A' that inverts f on uniformly random inputs of length $3n + \ell$ with probability $\varepsilon = \varepsilon(n)$, we can construct an adversary A with polynomially related running time such that, for independently sampled $sk \leftarrow \{0,1\}^n$ and $m \leftarrow \{0,1\}^{2n}$:

$$\Delta_A((E_{sk}(m), m), (E_{sk}(0^{2n}), m)) \geq \varepsilon - 2^{-n} \ .$$

If A' is efficient and ε is non-negligible, this suffices to break encryption (make sure you understand why). Given a ciphertext ct and a message m, $A(ct, m)$ runs $A'(ct, m)$ to obtain an alleged preimage (sk', m', r'). It outputs 1 if and only if $f(sk', m', r') = (ct, m)$, rejecting any malformed output.

Indeed, note that:

- When $ct \leftarrow E_{sk}(m; r)$, then A' gets exactly a random image under the function $(ct, m) = f(sk, m, r)$, and will thus successfully invert with probability ε.
- We'll show that when $ct \leftarrow E_{sk}(0^{2n}; r)$, it can invert with probability at most 2^{-n}. In what follows, $m \leftarrow \{0,1\}^{2n}$ and $sk \leftarrow \{0,1\}^n$ are sampled independently, and $ct \leftarrow E_{sk}(0^{2n})$ is generated using fresh randomness.

$$\begin{array}{lcll} \Pr\left[\begin{array}{l}(sk', m', r') \leftarrow A'(ct, m)\\ f(sk', m', r') = (ct, m)\end{array}\right] & = & \Pr\left[\begin{array}{l}(sk', m', r') \leftarrow A'(ct, m)\\ (E_{sk'}(m'; r'), m') = (ct, m)\end{array}\right] & \leq \\ \Pr\left[\ \exists sk' \in \{0,1\}^n : D_{sk'}(ct) = m\ \right] & \leq & \displaystyle\sum_{sk'} \Pr\left[D_{sk'}(ct) = m\right] & \leq \quad 2^n \cdot 2^{-2n} \ . \end{array}$$

The equality follows from the definition of f. The first inequality follows from correctness of decryption: a valid preimage yields a key sk' for which $D_{sk'}(ct) = m$. The second inequality is the union bound. For the last inequality, fix ct and sk'. Since m is uniform over $\{0,1\}^{2n}$ and independent of ct, the probability that $D_{sk'}(ct) = m$ is at most 2^{-2n}.

□

The Power of OWFs. We've argued that OWFs capture necessary properties for cryptography. The surprising part, and one of the great achievements of modern cryptography, is that *one-way functions are also sufficient* for everything we've seen so far, and for more that's to come, within the category of *secret-key crypto*.

Theorem 3.5 ([HILL99])**.** *OWFs imply PRGs!*

$$\boxed{\text{OWF}} \longrightarrow \boxed{\text{PRG}} \longrightarrow \boxed{\text{OEB Enc}} \longrightarrow \boxed{\text{Enc}}$$

The full proof of this amazing theorem is beyond the scope of this course. Nevertheless, starting in this lecture and continuing in the next, we'll develop some of the central concepts and tools that lead to this result. These will be interesting in their own right, and we'll use them to prove a weaker version.

4 Examples of OWFs

Before we go deeper into the above theorem, let's try to get a sense of what one-way functions could look like. Indeed, OWFs can be based on a wide variety of hard computational problems. For a comprehensive list of examples from different areas, see this essay by Boaz Barak, where he makes the point that if you throw a rock at random, you're likely to hit an OWF. Here, we'll give just three examples from different areas:

- **Factoring:**
  - Instance: $N = pq$, for two n-bit prime numbers p, q sampled at random.
  - Solution: p.

  Today the best (classical) algorithms for this problem run in time $\approx 2^{n^{\Omega(1)}}$. The problem can be solved efficiently by quantum algorithms.

- **Subset Sum:**
  - Instance: $x_1, \ldots, x_n, T$, where $x_i \leftarrow [2^n]$, and $T = \sum_{i \in I} x_i$ for a random subset $I \leftarrow 2^{[n]}$.
  - Solution: I' such that $T = \sum_{i \in I'} x_i$.

  Today the best (classical) algorithms for this problem run in time $\approx 2^{\Omega(n)}$.

- **Learning Parity with Noise:**
  - Instance: $(a_1, \langle a_1, s\rangle + e_1), \ldots, (a_{2n}, \langle a_{2n}, s\rangle + e_{2n})$, where $s \leftarrow \mathbb{F}_2^n$, $a_i \leftarrow \mathbb{F}_2^n$, e_i is 1 w.p. 0.1.
  - Solution: s (uniquely defined with overwhelming probability).

  Today the best (classical) algorithms for this problem run in time $\approx 2^{\widetilde{\Omega}(n)}$.

References

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.