Do PRGs exist? We don't know! More difficult problem than resolving P vs. NP!

However, it is not hard to see that if PRGs exist, then $P \neq NP$. [Try proving this yourself]

↳ What we can say is that if "one-way functions" (OWF) exist, then there exists a PRG that stretches the seed by 1 bit (e.g., λ-bit seed $\to$ $(\lambda+1)$-bit string)

function that is "easy" to compute but "hard" to invert

↳ will define more formally later in the course

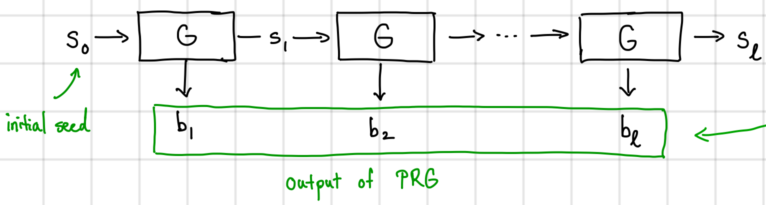
a PRG is an example of such a function

given $s \in \{0,1\}^\lambda$, evaluating $G(s) \in \{0,1\}^n$ is easy

given $G(s) \in \{0,1\}^n$ for random $s \in \{0,1\}^\lambda$, computing s is hard (why?)

But what if we want PRGs with longer stretch? For example, can we build PRGs with stretch $\ell(\lambda) = \text{poly}(\lambda)$ for arbitrary polynomials?

Blum-Micali PRG: Suppose $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$ is a secure PRG. We build a PRG with stretch $\ell(\lambda) = \text{poly}(\lambda)$ as follows:



denote the Blum-Micali construction $G^{(\ell)} : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$

Why is this constructing a secure PRG?

↳ Intuitively, if s_0 is uniformly random, then $G(s_0) = (b_1, s_1)$ is uniformly random so we can feed s_1 into the PRG and take b_1 as the first output bit of the PRG $\Rightarrow$ iterate until we have ℓ output bits

Theorem. If $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$ is a secure PRG, then the Blum-Micali generator $G^{(\ell)} : \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ is also a secure PRG for all $\ell = \text{poly}(\lambda)$.

Proof. Consider the following experiments:

Experiment H_0: Sample $s_0 \xleftarrow{R} \{0,1\}^\lambda$ and adversary is given $G^{(\ell)}(s_0)$

Experiment H_1: Sample $t \xleftarrow{R} \{0,1\}^{\ell(\lambda)}$ and adversary is given t

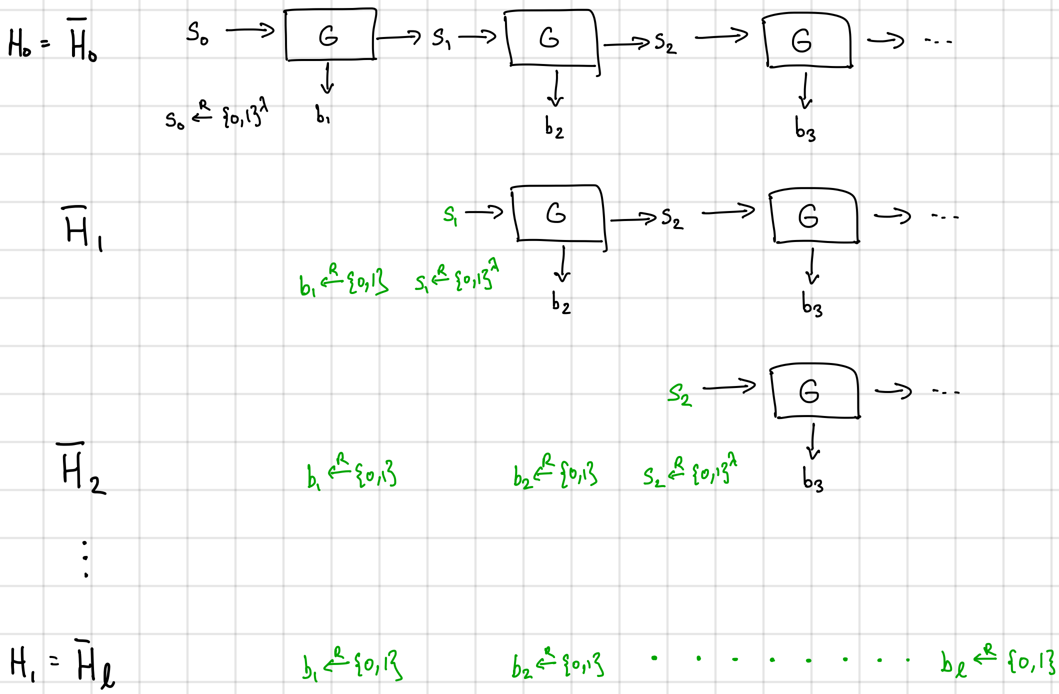
For an adversary A, define

$W_0 := \Pr[A \text{ outputs } 1 \text{ in } H_0]$

$W_1 := \Pr[A \text{ outputs } 1 \text{ in } H_1]$

Goal: Show that if G is secure, then for all efficient adversaries A, $|W_0 - W_1| = \text{negl}(\lambda)$.

We will use a "hybrid" argument. Specifically, we first define a sequence of intermediate experiments, where each adjacent pair of experiments is easy to reason about (i.e., directly reduces to security of G)



<u>Basic idea</u>: in experiment $\bar{H}_i$ the first i bits of output are generated <u>uniformly at random</u> while the remaining bits are generated using the Blum-Micali generator

In each experiment, adversary is given the sequence of bits $b_1 b_2 \cdots b_\ell$

Let A be an efficient distinguisher. Define $\bar{W}_i := \Pr[\text{A outputs 1 in experiment } \bar{H}_i]$

Then, $\text{PRGAdv}[A,G] = |W_0 - W_1|$

$= |\bar{W}_0 - \bar{W}_\ell|$ (by definition)

$= |\bar{W}_0 - \bar{W}_1 + \bar{W}_1 - \bar{W}_2 + \cdots + \bar{W}_{\ell-1} - \bar{W}_\ell|$

$\leq |\bar{W}_0 - \bar{W}_1| + |\bar{W}_1 - \bar{W}_2| + \cdots + |\bar{W}_{\ell-1} - \bar{W}_\ell|$ (by triangle equality)

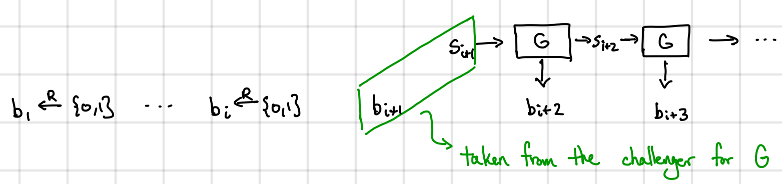
<u>Claim</u>. If G is a secure PRG, then for all efficient adversaries A, $|\bar{W}_i - \bar{W}_{i+1}| = \text{negl}(\lambda)$.

<u>Proof</u>. We will show the <u>contrapositive</u>: if A can distinguish experiments $\bar{H}_i$ and $\bar{H}_{i+1}$, then A can break pseudorandomness of G.

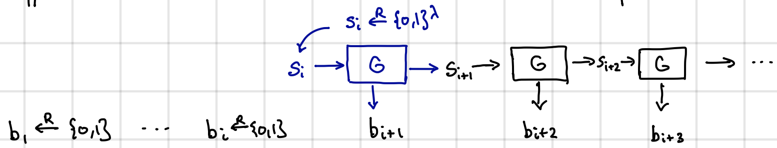
Suppose $|\bar{W}_i - \bar{W}_{i+1}| = \varepsilon$. We use A to build a distinguisher B for G. Algorithm B works as follows:

1. On input a string $z \in \{0,1\}^{\lambda+1}$, algorithm B parses z as (b_{i+1}, s_{i+1}) where $b_{i+1} \in \{0,1\}$ and $s_{i+1} \in \{0,1\}^\lambda$
2. Sample $b_1, \ldots, b_i \xleftarrow{R} \{0,1\}$.
3. Compute $b_{i+2}, \ldots, b_\ell$ using Blum-Micali with seed s_{i+1}. Give $b_1 \cdots b_\ell$ to A and output whatever A outputs.

In pictures:

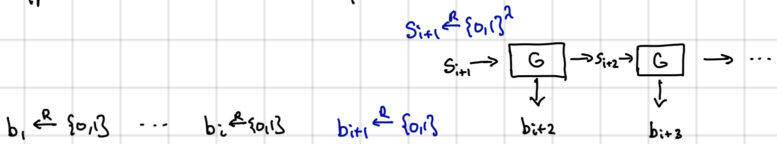


Two possibilities: 1. Suppose $z = G(s_i)$ for some $s_i \xleftarrow{R} \{0,1\}^\lambda$. Then, above picture looks like this:



In this case, $b_1, \dots, b_\ell$ is distributed exactly as in experiment $\overline{H}_i$ and so $\mathcal{A}$ outputs 1 with prob. $\overline{W}_i$

2. Suppose $z \xleftarrow{R} \{0,1\}^{\lambda+1}$. Then above picture looks like this:



In this case, $b_1, \dots, b_\ell$ is distributed exactly as in experiment $\overline{H}_{i+1}$ and so $\mathcal{A}$ outputs 1 with prob. $\overline{W}_{i+1}$

Very important to argue that $\mathcal{B}$ "simulates" the correct view for $\mathcal{A}$. Otherwise, behavior of $\mathcal{A}$ is unknown!

Thus, $\mathrm{PRGAdv}[\mathcal{B}, G] = |\overline{W}_i - \overline{W}_{i+1}|$
$= \varepsilon$ since $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs

Since $\mathcal{B}$ is efficient (assuming $\mathcal{A}$ is efficient), by security of G, $\mathrm{PRGAdv}[\mathcal{B}, G] = \mathrm{negl}(\lambda)$. Thus, $\varepsilon = |p_i - p_{i+1}| = \mathrm{negl}(\lambda)$, and the claim follows. ■

To complete the proof of the main theorem, we have that

$$\begin{aligned} |\overline{W}_0 - \overline{W}_\ell| &\le |\overline{W}_0 - \overline{W}_1| + \cdots + |\overline{W}_{\ell-1} - \overline{W}_\ell| \\ &\le \ell \cdot \mathrm{negl}(\lambda) \\ &= \mathrm{negl}(\lambda) \quad \text{since } \ell = \mathrm{poly}(\lambda). \end{aligned}$$

<u>Proof strategy recap</u>: 1. Hybrid arguments: to argue indistinguishability of a pair of distributions, begin by identifying a <u>simple</u> set of intermediate distributions, and argue that each pair of adjacent distributions is indistinguishable

2. Security reduction (proof by contrapositive): To show a statement of the form "If X is secure, then Y is secure," show instead the the statement "If Y is not secure, then X is not secure." In the proof, show that if there exists an adversary for Y (i.e. Y is not secure), then there exists an adversary for X.

↳ When constructing this adversary, it is important to show that it simulates the <u>correct distribution</u> of inputs to the underlying adversary (i.e., this is essentially showing correctness of the <u>reduction algorithm</u>)

Stream ciphers in practice:

- Salsa20 (2005) ⇝ ChaCha (2008)
  - ↳ core design maps 256-bit key, 64-bit nonce, 64-bit counter onto a 512-bit output

    enables using same key (and different nonces) to encrypt <u>multiple</u> messages (will discuss later)

    allows <u>random</u> access into the stream

    Design is more complex:
    - relies on a sequence of rounds
    - each round consists of 32-bit additions, xors, and bit-shifts
  - ↳ very fast even in software (4-14 CPU cycles / output byte) — used to encrypt TLS traffic between Android and Google services