Lecture 2: From PRGs to PRFs

4/1/21

Plan

Brief recap

OWFs

PRG security

PRGs

1 bit stretch → arbitrary stretch

Security proof & hybrid arguments ← perhaps the most important proof technique in modern crypto!

PRFs

Definition review

PRGs → PRFs

Bonus content!

A simple hybrid argument for a widely used scheme

Logistics:
- HW1 out, please start early!
- OH Wed-Sat, 3x on Fri, Saba covering for Riad this week

  Are the times ok?
- Don't forget to sign up for Piazza/Gradescope

Recap: OWFs → symmetric crypto

$$
\begin{tikzcd}[column sep=huge]
\text{OWFs} \arrow[r, bend left, "\substack{\text{Tuesday} \\ \text{Goldreich-Levin}}"] & \text{PRGs} \arrow[l, bend left, "\text{by definition}"] \arrow[r, bend left, "\substack{\text{Today!} \\ \boxed{\text{Blum-Micali GGM}}}"] & \text{PRFs} \arrow[l, bend left, "\text{counter mode}"] \arrow[r, bend left, "\substack{\text{Feistel networks/} \\ \text{Luby-Rackoff}}"] & \text{PRPs} \arrow[l, bend left, "\substack{\text{switching} \\ \text{lemma}}"] \arrow[r, "\substack{\text{MAC from PRF} \\ + \\ \text{Enc then MAC}}"] & \substack{\text{Auth} \\ \text{Enc}}
\end{tikzcd}
$$

Def: A PRG $G: \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ is a deterministic poly-time algorithm. It is secure if for all PPT adversaries A:

$$\left| \Pr\left[s \xleftarrow{R} \{0,1\}^\lambda : A(G(s)) = 1 \right] - \Pr\left[t \xleftarrow{R} \{0,1\}^{\ell(\lambda)} : A(t) = 1 \right] \right| \leq \mathrm{negl}(\lambda)$$

AHA! $\{ s \xleftarrow{R} \{0,1\}^\lambda : G(s) \} \approx_c \{ t \xleftarrow{R} \{0,1\}^{\ell(\lambda)} : t \}$

Building Better PRGs

Last time: OWF → PRG with 1 bit stretch
(OWP) $G: \{0,1\}^n \to \{0,1\}^{n+1}$

Today: PRG with 1 bit stretch → PRG with arbitrary (polynomial) stretch
$G: \{0,1\}^n \to \{0,1\}^{\ell(n)}$

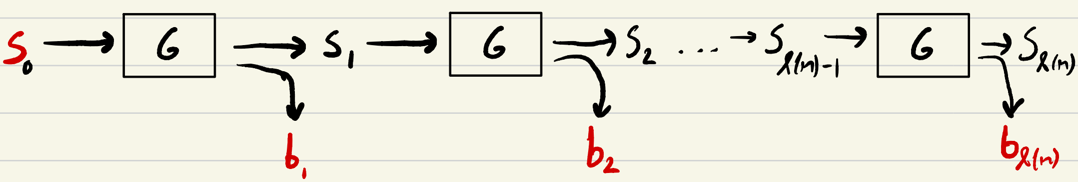
The Blum-Micali PRG

Let $G: \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG,

and let $\ell(n)$ be a polynomial.

We construct $G': \{0,1\}^n \to \{0,1\}^{\ell(n)}$

Observation: We can chain PRG evals together, and each time we get one extra pseudorandom bit left over!



Algorithm: on input $s \in \{0,1\}^n$:

- set $s_0 \leftarrow s$
- for $i = 1, 2, \dots, \ell(n)$: $(s_i, b_i) \leftarrow G(s_{i-1})$
- output $b_1, b_2, \dots, b_{\ell(n)}$

Theorem: G is a secure PRG $\to G'$ is a secure PRG

- G' is polytime? let $t(n)$, $t'(n)$ be time taken by G, G'

$$t'(n) = \ell(n) \cdot t(n) + O(\ell(n)) \quad \checkmark$$

Product of polynomials is always a polynomial

- G' is secure?

Intuitively, all the s_i, b_i look random by security of G, so the output $b_1, \dots, b_{\ell(n)}$ should look random too. How to formalize?

Need to show $\underline{\{ s \xleftarrow{R} \{0,1\}^n : G'(s) \}} \approx_c \{ y \xleftarrow{R} \{0,1\}^{\ell(n)} \}$

Given $\{ s \xleftarrow{R} \{0,1\}^n : G(s) \} \approx_c \{ y \xleftarrow{R} \{0,1\}^{n+1} \}$

from definition of PRG

So what really is this?

$$\{ s \xleftarrow{R} \{0,1\}^n : G'(s) \} = \left\{ \begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ (s_1, b_1) \leftarrow G(s) \\ (s_2, b_2) \leftarrow G(s_1) \\ \quad \vdots \\ (s_\ell, b_\ell) \leftarrow G(s_{\ell-1}) \end{array} : \underbrace{b_1, \dots, b_n}_{} \overset{G'(s)}{} \right\}$$

just the description of the algorithm $G(s)$

issue: definition of PRG gives us that $G(s)$ looks random when $s \xleftarrow{R} \{0,1\}^n$. But here we usually have that $s_i \leftarrow G(s_{i-1})$, which is something different. how do we use the definition to formally prove security?

Solution: we will prove security by applying the definition to one PRG use at a time!

$$\underbrace{\left\{\begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ (s_1, b_1) \leftarrow G(s) \\ (s_2, b_2) \leftarrow G(s_1) \\ \vdots \\ (s_\ell, b_\ell) \leftarrow G(s_{\ell-1}) \end{array}\right\}}_{D_0} \approx_c \underbrace{\left\{\begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ (s_1, b_1) \xleftarrow{R} \{0,1\}^{n+1} \\ (s_2, b_2) \leftarrow G(s_1) \\ \vdots \\ (s_\ell, b_\ell) \leftarrow G(s_{\ell-1}) \end{array}\right\}}_{D_1} \approx_c \underbrace{\left\{\begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ (s_1, b_1) \xleftarrow{R} \{0,1\}^{n+1} \\ (s_2, b_2) \xleftarrow{R} \{0,1\}^{n+1} \\ \vdots \\ (s_\ell, b_\ell) \leftarrow G(s_{\ell-1}) \end{array}\right\}}_{D_2}$$

$$\vdots$$

intuitively, using $\{s \xleftarrow{R} \{0,1\}^n : G(s)\} \approx_c \{y \leftarrow \{0,1\}^{n+1} : y\}$
(will formalize soon)

$$\approx_c \underbrace{\left\{\begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ (s_1, b_1) \xleftarrow{R} \{0,1\}^{n+1} \\ (s_2, b_2) \xleftarrow{R} \{0,1\}^{n+1} \\ \vdots \\ (s_\ell, b_\ell) \xleftarrow{R} \{0,1\}^{n+1} \end{array}\right\}}_{D_\ell}$$

Now $b_1, \dots, b_\ell \xleftarrow{R} \{0,1\}^\ell$

each distribution can only be distinguished from the next with at most negligible probability (smaller than any polynomial). So adding polynomially many of them will still be negligible.

Security: $D_0 \approx_c D_2 \approx_c \cdots \approx_c D_\ell \rightarrow D_0 \approx_c D_\ell$

This technique is called a **hybrid argument** and is used <u>everywhere</u> in crypto

Hybrids in pictures:

$$D_0: \quad \begin{tikzcd} s_0 \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_1 \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_2 \cdots \arrow[r] & s_{\ell(n)-1} \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_{\ell(n)} \\ & b_1 & & b_2 & & & b_{\ell(n)} & \end{tikzcd}$$

$$\approx_c$$

$$D_1: \quad \begin{tikzcd} s_0 & s_1, b_1 \xleftarrow{R} \{0,1\}^{n+1} \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_2 \cdots \arrow[r] & s_{\ell(n)-1} \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_{\ell(n)} \\ & & b_2 & & & b_{\ell(n)} & \end{tikzcd}$$

$$\approx_c$$

$$D_2: \quad \begin{tikzcd} s_0 & s_1, b_1 \xleftarrow{R} \{0,1\}^{n+1} & s_2, b_2 \xleftarrow{R} \{0,1\}^{n+1} & \cdots \arrow[r] & s_{\ell(n)-1} \arrow[r] & \boxed{G} \arrow[r] \arrow[d] & s_{\ell(n)} \\ & & & & & b_{\ell(n)} & \end{tikzcd}$$

$$\vdots$$

$$D_\ell: \quad s_0 \qquad s_1, b_1 \xleftarrow{R} \{0,1\}^{n+1} \qquad s_2, b_2 \xleftarrow{R} \{0,1\}^{n+1} \qquad \cdots \qquad s_{\ell(n)}, b_{\ell(n)} \xleftarrow{R} \{0,1\}^{n+1}$$

One last important piece: how to prove neighboring distributions are computationally indistinguishable?

Wait, didn't we prove that by putting $\approx_c$ between them and using the definition?

Kinda, but that's more for intuition and isn't really a proof. See how it can go wrong in a different example

$$\left\{ \begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ s' \leftarrow G(s) \end{array} : s, s' \right\} \not\approx_c \left\{ \begin{array}{l} s \xleftarrow{R} \{0,1\}^n \\ s' \xleftarrow{R} \{0,1\}^{n+1} \end{array} : s, s' \right\}$$

These two distributions are clearly different b/c the seed s appears in both, so a distinguishing adversary can just check if $G(s) = s'$.

Proof that D_i, D_{i+1} are indistinguishable:

Suppose for the sake of contradiction that there exists an adversary A who can distinguish between D_i and D_{i+1}. We will use A to build another adversary B who breaks the security of G.

Recall: D_i : i random bits and then PRG output bits
D_{i+1} : $i+1$ random bits and then PRG output bits

A distinguishes these

B is given $z \in \{0,1\}^{n+1}$ and needs to decide if
1) $z \xleftarrow{R} \{0,1\}^{n+1}$
2) $s \xleftarrow{R} \{0,1\}^n$; $z \leftarrow G(s)$

B's strategy:

1) parse z as (s_{i+1}, b_{i+1}) where $s_{i+1} \in \{0,1\}^n$ $b_{i+1} \in \{0,1\}$

2) choose random $b_1, \ldots, b_i \xleftarrow{R} \{0,1\}$

3) set $s_{i+2}, b_{i+2} \leftarrow G(s_{i+1})$, ..., $s_\ell, b_\ell \leftarrow G(s_{\ell-1})$

4) set $y \leftarrow b_1, \ldots, b_\ell$

5) run $A(y)$ and output whatever A outputs.

Observe: if $z \xleftarrow{R} \{0,1\}^{n+1}$ A is receiving a sample from D_{i+1}
if $z \leftarrow G(s)$ A is receiving a sample from D_i

↳ B determines if z is random or pseudorandom with the same (non-negligible) advantage that A distinguishes D_i & D_{i+1}!

This contradicts the PRG security of G, so such an A cannot exist! $\blacksquare$

Note: See Section 3.4 of Boneh-Shoup textbook for more mathematical details.

PRFs

Definition: Pseudorandom function (PRF)

A PRF $F: \overset{\text{Key space}}{K} \times \overset{\text{domain}}{X} \to \overset{\text{range}}{Y}$ is secure if for all PPT Adversaries A:

$$\text{PRFAdv}[A, F] = |\Pr[W_0] - \Pr[W_1]| < \text{negl}(\lambda)$$

where for $b \in \{0,1\}$, W_b is defined as A's output being 1 in the following experiment

Adversary | **Challenger**

Challenger: if $b=0$: $k \xleftarrow{R} K$
if $b=1$: $f \xleftarrow{R} \text{Funcs}[X, Y]$ ← set of all functions from X to Y

Adversary → Challenger: $x \in X$

Challenger: if $b=0$: $y = F(k, x)$
if $b=1$: $y = f(x)$

Challenger → Adversary: y

Adversary outputs $b' \in \{0,1\}$

Intuition: Adversary can't distinguish outputs of a PRF from outputs of a truly random function.

Related: Pseudorandom Permutation (PRP): Same as PRF, but $F(k, \cdot)$ is also a permutation (a bijective function)

Recall from CS255 that PRFs and PRPs are the abstractions we usually use for block ciphers

PRGs → PRFs

We will show how to go from a <u>length-doubling PRG</u> to a PRF

$$G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$$

Notation: $G(s) \to (s_0, s_1) = (G_0(s), G_1(s))$

Observation: G can be viewed as a PRF with a 1-bit domain

$$F : \underbrace{\{0,1\}^\lambda}_{\text{Key}} \times \underbrace{\{0,1\}}_{\text{1-bit domain}} \to \underbrace{\{0,1\}^\lambda}_{\text{range}}$$

$$F(\underset{\text{Key}}{s}, \underset{\text{input bit}}{0}) = G_0(s) \qquad\qquad F(s,1) = G_1(s)$$

But a PRF with a 1-bit domain isn't really that useful for us. E.g., we often use block ciphers like AES with a 128-bit domain.

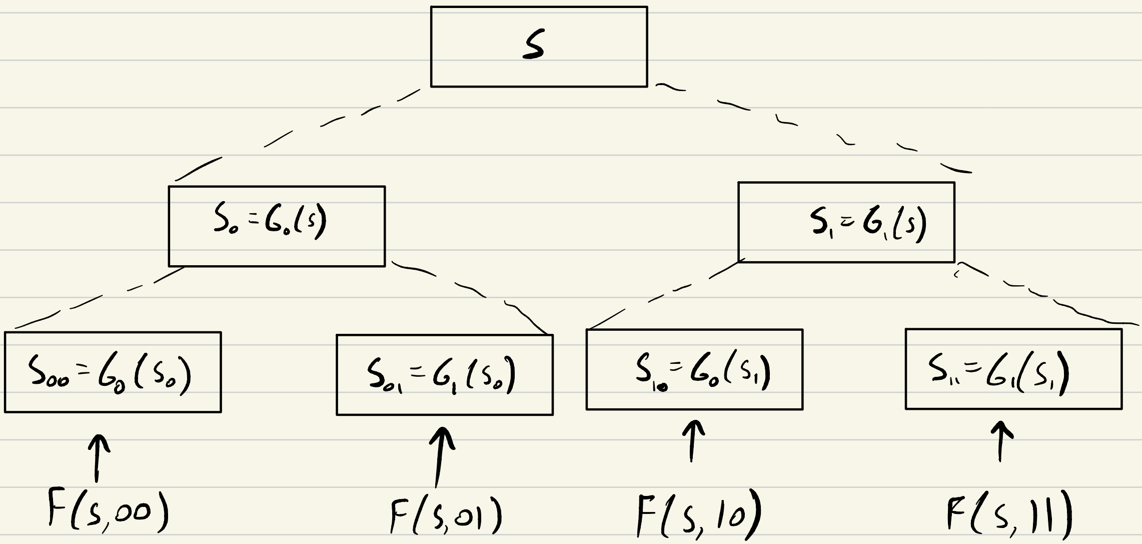
How do we generalize to an arbitrary domain $X_\lambda = \{0,1\}^{n(\lambda)}$?

Recursion!

Goldreich-Goldwasser-Micali (GGM) construction:

$$F(s, \underbrace{x_1 x_2 \ldots x_n}_{\text{bit decomposition of input}}) = G_{x_n}(G_{x_{n-1}}(\cdots G_{x_1}(s) \cdots))$$

Picture (for $n=2$)



Tree continues for as many layers as there are bits in the domain of the PRF F.

Is computing F poly-time? Yes! only need to evaluate path to leaf for chosen input $x \in X$. Thus only $\log(|X|)$ PRG calls needed.

How do we prove this is a Secure PRF?

- Assume G is a secure PRG
- Show that if an adversary has non-negligible advantage against the PRF, then there exists an adversary with non-negligible advantage against the PRG → contradiction!

This step uses a hybrid argument. see Boneh-Shoup section 4.6 for the details if interested.

Bonus Content!

How do we often use public key encryption?

Given PK enc scheme (PK.Enc, PK.Dec)
and Symmetric enc scheme (Sym.Enc, Sym.Dec)

$\text{Enc}(pk, m)$: $\quad k \xleftarrow{\$} \mathcal{K}$
$\quad u \leftarrow \text{PK.Enc}(pk, k)$
$\quad v \leftarrow \text{Sym.Enc}(k, m)$
$\quad$ return $ct \leftarrow (u, v)$

$\text{Dec}(sk, ct)$: $\quad (u, v) \leftarrow ct$
$\quad k \leftarrow \text{PK.Dec}(sk, u)$
$\quad m \leftarrow \text{Sym.Dec}(k, v)$
$\quad$ return m

Recall we do this b/c PK crypto is more expensive than symmetric crypto.

This way we only use PK crypto for a small key and encrypt the long msg with symmetric crypto.

Confusingly, this is sometimes referred to as Hybrid encryption

How do we prove that this approach is secure? A hybrid argument!

Recall: Security for Encryption (one-time CPA security)

$$\underbrace{\{\text{Enc}(pk, m_1)\}}_{\{\text{PK.Enc}(pk, k),\ \text{Sym.Enc}(k, m)\}} \approx_c \{\text{Enc}(pk, m_2)\}$$

Same idea applies for many-time security.

1. Need to invoke security of both PK and sym encryption.
2. At first glance, can't use def of sym enc b/c the key appears in the distribution, and security relies on key being hidden.

$$\{Enc(PK, m_1)\}$$

$$= \{PK.Enc(pk, K), Sym.Enc(K, m)\}$$

$$\approx_c \{PK.Enc(pk, 0), Sym.Enc(K, m_1)\}$$ by security of PK

$$\approx_c \{PK.Enc(pk, 0), Sym.Enc(K, m_2)\}$$ by security of Sym

$$\approx_c \{PK.Enc(pk, K), Sym.Enc(K, m_2)\}$$ by security of PK

$$= \{Enc(PK, m_2)\}$$

Good exercise to formalize these steps. Feel free to bring this up at office hours!

Key takeaway:

Often in crypto we want to use the security of multiple primitives to prove some scheme secure (or use the security of the same primitive multiple times). Hybrid arguments allow us to break the problem into small pieces and use the security of the primitives used in a construction one at a time.