0368.4162 Foundations of Cryptography November 23, 2017

Lecture 5: Pseudorandom Functions

Lecturer: Nir Bitansky *Scribes: Yoav Kaempfer, Lior Solodkin*

1 Previously on Foundations of Crypto

We've seen that any one-way function (OWF) has a (randomized) hardcore bit (HCB), and saw how to use such a HCB to construct a pseudorandom generator (PRG) out of one-way permutations. As already mentioned last lecture, the more general theorem that OWFs imply PRGs is also true, but requires additional techniques that are not within the scope of this course.

So far we've seen that starting from OWFs, we can obtain PRGs and encryption for messages of arbitrary polynomial length. We now aim to achieve additional cryptographic goals based on these tools, and see how far they could get us.

$$\begin{tikzcd}
 & & \boxed{?} \\
\boxed{\text{OWF}} \arrow[r] & \boxed{\text{PRG}} \arrow[ur] \arrow[r] \arrow[dr] & \boxed{\text{Enc}} \\
 & & \boxed{?}
\end{tikzcd}$$

2 Pseudorandom Functions

Our first step will be to construct a much stronger form of PRG, which we will call a *pseudorandom function* (PRF). Let us start by motivating this object, within the familiar context of encryption.

A Motivating Question: Encrypting Multiple Messages. We know that assuming OWFs, we can encrypt messages of arbitrary polynomial length $\ell(n) = n^{O(1)}$ (using a key of fixed size n). Our security definitions, however, only dealt with attackers that see a single (perhaps long) ciphertext.

Can we securely encrypt multiple messages?

There are several ways of defining what security for multiple messages actually means. For now, we will consider the following natural definition known as security against known plaintext attacks (KPA).[1] Also, for simplicity we will restrict attention to single bit messages. This is w.l.o.g in the sense that to encrypt longer messages, we can encrypt their bits separately (although this may cost us in efficiency).

Definition 2.1 (Multi-Message Encryption (against Known Plaintext Attacks))**.** *A bit encryption scheme (E, D) is multi-message secure if for any polynomial $\ell(\cdot)$,*

$$\{E_{sk}(x_1), \dots, E_{sk}(x_\ell)\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}} \approx_c \{E_{sk}(y_1), \dots, E_{sk}(y_\ell)\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell(n)} \\ y\in\{0,1\}^{\ell(n)}}} ,$$

where in the above $sk \leftarrow \{0,1\}^n$.

[1] In a nutshell, in a known plaintext attack, the encrypted messages are known ahead of time. Later, we shall define the notion of chosen plaintext attacks, where the attacker may adaptively request to see different encrypted messages, depending on all encryptions it had seen so far.

Apriori we may hope that any encryption scheme that's secure for a single message would also be secure if we reuse it for two messages. However, we've already seen that this is not true in general (for instance, in one-time pad). Still, we may hope to construct a designated scheme that will be secure for multiple messages. In fact, in some sense, we've already did.

Stateful Encryption. Our construction of encryption for long messages from one-extra-bit encryption actually already supports many messages to some extent. Specifically, it had a chain structure where we could always encrypt (and decrypt) an additional bit m_{i+1} provided that the encryptor (and decryptor) has the current secret key sk_i. Such *stateful* encryption schemes are commonly known as *stream ciphers* or *state ciphers*. The obvious caveat of such encryption schemes is that they require the parties to remain synchronized. In particular, if some messages are dropped, then we can no longer guarantee consistent decryption. There are solutions that can *resynchronize* after a few failures, but in general this is a problem.

Publicly Synchronizing. One naïve way to solve the synchronization problem is for the encryptor to send his state online. This of course cannot work in general — the state may contain secret randomness (e.g., the next secret key) that will compromise security. However, using PRGs it seems that we can actually construct an encryption scheme that only requires keeping a public state, and in fact, a pretty short one.

The construction is simple: The encryptor would keep a counter of the number of messages it encrypted so far. When it wants to encrypt the ith bit, it will first expand its n-bit random secret key s using a PRG G and use the ith output bit $G(s)_i$ as a one-time pad. Specifically, it will send $i, G(s)_i \oplus m_i$ as the encryption. (Recall that we can indeed extend the output of G as much as we want to any polynomial number of bits.)

In this scheme, there is no issue of synchronization, but the scheme still has some undesired properties. First, the encryptor has to maintain a state (the counter i). Furthermore, the complexity of encryption/decryption potentially grows with the number of encrypted messages — this is certainly the case if we use the chain-style PRG extension we've previously seen.

The Solution: PRFs. Both of the above problems can be solved using PRFs, which intuitively can be viewed as a PRG with an *exponential* number of output bits where the ith output bit $G(s)_i$ can be computed in fixed polynomial time (independent of i). There are several conceivable ways to formalize this. We will consider a relatively strong definition which requires that the function $i \longmapsto G(s)_i$ is indistinguishable from a truly random function.

Definition 2.2 (Pseudorandom Function (PRF))**.** *A pseudorandom function is a polynomial-time function F that takes as input a seed $s \in \{0,1\}^n$ and an index $i \in \{0,1\}^n \cong [2^n]$ and outputs a bit, such that for any n.u. PPT $A = \{A_n\}_{n\in\mathbb{N}}$ there exists a negligible μ such that for all $n \in \mathbb{N}$:*

$$\left|\Pr\left[A_n^{F_s(\cdot)} = 1 \;\middle|\; s \leftarrow \{0,1\}^n\right] - \Pr\left[A_n^{R(\cdot)} = 1 \;\middle|\; R \leftarrow \{0,1\}^{\{0,1\}^n}\right]\right| \leq \mu(n) \ .$$

In this definition, A_n is an oracle-aided algorithm (it does not view the function at its entirety, but can make a polynomial number of arbitrary queries). The notation $\{0,1\}^{\{0,1\}^n}$ stands for the set of all functions mapping $\{0,1\}^n$ to $\{0,1\}$. In other words, R is simply a random function, that for each input string returns a random independent bit.[2]

Encrypting Multiple Messages Using PRFs. Given PRFs, we can now follow the same rational as before, but completely lose the state, and in particular, encrypt in fixed time (independently of the number of messages encrypted so far). The scheme is given by:

- $E_{sk}(m)$:
  - Interpret sk as a seed for a PRF.

[2]Note that PRFs indeed strengthen the notion of PRGs. In particular, any PRF F can be used to construct a PRG G (of arbitrary polynomial length), where the ith output bit of $G(s)$ is simply $F_s(i)$.

  - Sample a random $r \leftarrow \{0,1\}^n$ and output $r, F_{sk}(r) \oplus m$.

- $D_{sk}(r, v)$:
  - Output $v \oplus F_{sk}(r)$.

It's easy to verify the correctness of the scheme. Let's sketch the proof of security.

Multi-Message Security, Sketch. Notice that we can describe our encryption algorithm E_{sk} as an oracle-aided algorithm $\mathcal{E}^{F_{sk}}$ that uses F_{sk} as a black box. Now, we can consider an alternative (mental) experiment, where instead of using F_{sk} we use a truly random oracle R.

Claim 2.3. *For any polynomial ℓ,*

$$\left\{\mathcal{E}^{F_{sk}}(x_1), \ldots, \mathcal{E}^{F_{sk}}(x_\ell) \mid sk \leftarrow \{0,1\}^n\right\}_{\substack{n\in\mathbb{N}\\ x\in\{0,1\}^{\ell(n)}}} \approx_c \left\{\mathcal{E}^{R}(x_1), \ldots, \mathcal{E}^{R}(x_\ell) \;\middle|\; R \leftarrow \{0,1\}^{\{0,1\}^n}\right\}_{\substack{n\in\mathbb{N}\\ x\in\{0,1\}^{\ell(n)}}} .$$

This claim follows directly from the security of the PRF. Indeed, given a non-uniform PPT A that distinguishes between the above two ensembles (for infinitely many n and $x \in \{0,1\}^{\ell(n)}$), we can construct a distinguisher $B^{(\cdot)}$ for the pseudorandom function. The distinguisher $B^{(\cdot)}$ simply emulates $\mathcal{E}^{(\cdot)}$ to encrypt the bits of x, and answer the oracle calls that $\mathcal{E}^{(\cdot)}$ makes using its own oracle. B distinguishes a PRF from a random function with exactly the same advantage that A distinguishes encryptions relative to F_{sk} from encryptions relative to R.

This means that to finish the proof, we just need to show that the scheme is secure when instantiated with a random function. This is actually true in a statistical (rather than computational sense).

Claim 2.4.

$$\left\{\mathcal{E}^{R}(x_1), \ldots, \mathcal{E}^{R}(x_\ell)\right\}_{\substack{n\in\mathbb{N}\\ x\in\{0,1\}^{\ell(n)}\\ y\in\{0,1\}^{\ell(n)}}} \approx_s \left\{\mathcal{E}^{R}(y_1), \ldots, \mathcal{E}^{R}(y_\ell)\right\}_{\substack{n\in\mathbb{N}\\ x\in\{0,1\}^{\ell(n)}\\ y\in\{0,1\}^{\ell(n)}}} .$$

To see that this is true, notice that as long as the random values $r_1, \ldots, r_\ell$ chosen by $\mathcal{E}$ are distinct, then the produced encryptions $(r_1, R(r_1) \oplus x_1), \ldots, (r_\ell, R(r_\ell) \oplus x_\ell)$ are uniformly random, independently of the underlying message x. Thus, it suffices to bound the probability that the random strings r_i are not distinct. There are at most $\binom{\ell}{2}$ such potential collisions, each occurring with probability 2^{-n}. Overall, by a union bound, the probability that these strings are not distinct is at most $2^{-n}\ell^2$, which is negligible. □

Note on Randomized Encryption. Note that the constructed encryption scheme is randomized. In fact, any stateless encryption scheme for more than a single message must be randomized (think why).

3 The GGM Construction

In '86 Goldreich, Goldwasser, and Micali gave a beautiful construction of a PRF from any PRG.

Theorem 3.1 ([GGM86])**.** *If there exist PRGs, then there exist PRFs.*

Corollary 3.2. *Assuming OWFs, there exist PRFs.*

Proof. The construction is based on a recurring theme in cryptographic constructions — replacing a *chain* by a *tree*, which has the advantage that the number of nodes we can reach is exponential in the depth (instead of linear). Specifically, let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling PRG. For a seed s, we'll denote by $G_0(s)$ and $G_1(s)$ the two n-bit parts of the output $G(s)$.

We define:

$$F_s(x) = G_{x_n}(\cdots G_{x_2}(G_{x_1}(s))\cdots) \; .$$

We will now analyze the security of the construction. We will rely on the following claim.

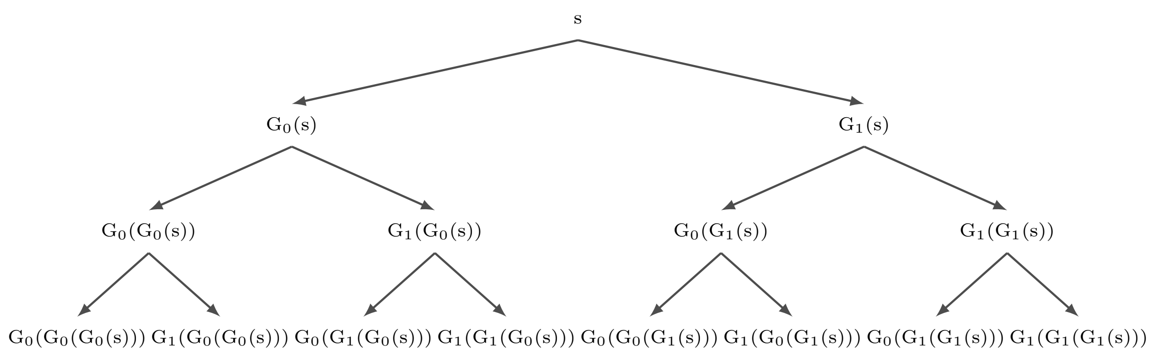

Figure 1: A GGM Tree for inputs of length $n = 3$. To compute $F_s(x_1x_2x_3)$ follow the corresponding path and output the resulting leaf.

Claim 3.3. *For any polynomial $q = q(n)$,*

$$\left\{U_{2n}^{(1)}, \dots, U_{2n}^{(q)}\right\}_{n\in\mathbb{N}} \approx_c \left\{G(U_n^{(1)}), \dots, G(U_n^{(q)})\right\}_{n\in\mathbb{N}} .$$

The claim can be proved by a hybrid argument (you proved a more general claim in the homework).

We turn to prove the security of our constructed function F. Let A be an adversary that distinguishes the function F_s (for a random s) from a random function R, with advantage $\varepsilon = \varepsilon(n)$, and assume it makes at most $q = q(n)$ queries to its oracle. We will construct a corresponding distinguisher B for the two distribution ensembles in Claim 3.3.

We consider the following labeling of the nodes $\left\{x_1 \dots x_i \;\middle|\; \begin{array}{c} 0 \le i \le n \\ x_1 \dots x_i \in \{0,1\}^i \end{array}\right\}$ in the full binary tree. The root of the tree ε (at level zero) is labeled by the random seed $L_\varepsilon := s$. An internal node $x_1 \dots x_i$ (at level i) is recursively labeled by $G_{x_i}(L_{x_1 \dots x_{i-1}})$. Note that each leaf $x_1 \dots x_n$ will be labeled by $L_{x_1 \dots x_n} := F_s(x_1 \dots x_n)$.

We now consider $n+1$ hybrid experiments $H_0, \dots, H_n$ where we gradually change the labels as follows. In H_i, all the labels $L_{x_1 \dots x_i}$ for nodes at level i are sampled independently at random. Then, the rest of the labels down the tree are chosen according to the same recursive rule as before (the labels at levels $j < i$ are ignored). In each of these hybrid experiments, the adversary's queries are answered according to the leaf labels.

Note that H_0 corresponds exactly to the case that the labels are given by the PRF $L_{x_1 \dots x_n} = F_s(x_1 \dots x_n)$, whereas H_n corresponds to the case that the labels are chosen by a truly random function $L_{x_1 \dots x_n} = R(x_1 \dots x_n)$. Then, there exists an $i \in [n]$ such that A distinguishes H_{i-1} from H_i with advantage ε/n.

We note the following difference between the two hybrids:

- In H_{i-1}, all the labels in layer i are chosen pseudorandomly (by applying G to their parent labels).
- In H_i, all the labels in layer i are chosen completely at random.

We now construct the distinguisher B. Recall that B obtains t samples $Y^{(1)}, \dots, Y^{(t)}$ and has to efficiently decide whether they where sampled at random or pseudorandomly. Intuitively, we'd like to use these samples as the labels of ith level in the tree. However, the size of the ith layer could very well be exponential. So we may not have enough samples to do so, and in any case certainly cannot sample all labels efficiently.

The point is that the adversary A, doesn't really look at all the leafs, but on at most t of them, so most of the labels are irrelevant. Specifically, we will compute the labels in a lazy fashion.

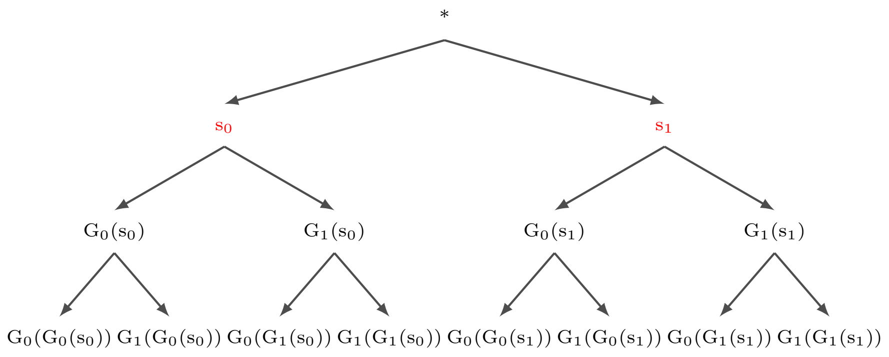
*
s0
s1
G0(s0)
G1(s0)
G0(s1)
G1(s1)
G0(G0(s0)) G1(G0(s0)) G0(G1(s0)) G1(G1(s0)) G0(G0(s1)) G1(G0(s1)) G0(G1(s1)) G1(G1(s1))

Figure 2: H_i illustrated for $i = 1$

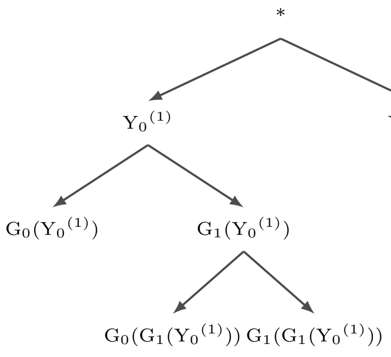

Figure 3: Lazy labeling using the list $Y^{(1)}, ..., Y^{(q)}$ for level i. Paths in the tree are only labeled per As queries.

Algorithm $B(Y^{(1)}, \dots, Y^{(q)})$**:**

- Initialize $\mathcal{L} = \emptyset$ (the set of i-level labels encountered so far).
- Initialize a counter $c = 0$ (the number of samples used so far from the list $Y^{(1)}, \dots, Y^{(q)}$).
- Emulate A, and when it makes a query x act as follows:
  1. If $\mathcal{L}$ contains a pair $(x_1 \dots x_i, L_{x_1 \dots x_i})$, compute the labels down the path from $x_1 \dots x_i$ to $x_1 \dots x_n$ according to the recursive rule, and return the corresponding leaf.
  2. If $\mathcal{L}$ has no such element:
     - Use the next sample $Y^{(c+1)} = Y_0^{(c+1)} Y_1^{(c+1)}$ to add such an element and a sibling $(x_1 \dots x_{i-1}0, Y_0^{(c+1)}), (x_1 \dots x_{i-1}1, Y_1^{(c+1)})$ to the list $\mathcal{L}$.
     - Increase the counter c by one.
     - Return to step (1) (now we can compute the corresponding leaf).
- At the end output whatever A outputs.

It is left to see that if the samples $Y^{(1)}, \dots, Y^{(q)}$ are pseudorandom, then A's emulated view is exactly as in H_{i-1}, whereas if they are random, then it's exactly as in H_i. Thus, B manages to distinguish the two cases with advantage ε/n. This means that $\epsilon(n) = n^{-\omega(1)}$. □

References

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.