University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 13

1 Completing the Proof for the PRG Construction

Recall the PRG construction beginning with any one-way permutation f with hard-core bit h: we define $G(x) = f(x) \circ h(x)$. We complete the proof in lecture today; see the lecture notes for Lecture 12 for the full proof.

2 Increasing the Expansion

In the previous lecture we discussed how to convert a PRG that expands its input by a single bit to a PRG that expans its input by a larger amount. For completeness, we state the result of this transformation — applied to our particular construction — here. Recall that the idea is to form $H(x) = G(\cdots G(x) \cdots)$, where G can be iterated as many times as desired. Also, we stated that it is not necessary to apply G to the *entire* string at each intermediate stage; instead, we can apply G to only the first, say, k bits of every intermediate string. Applying this methodology to our construction for G, above, gives:

$$H(x) = f^{\ell}(x) \circ h(f^{\ell-1}(x)) \circ h(f^{\ell-2}(x)) \circ \cdots \circ h(f(x)) \circ h(x),$$

where H is now a PRG that increases its input by ℓ bits.

3 PRGs and One-Way Functions

We have discussed in class the construction of a PRG from a one-way permutation, showing that PRGs exist if one-way permutations exist. Actually, we can state a more general result.

Theorem 1 *PRGs exist if and only if one-way* functions *exist.*

One direction of the theorem is easy (I would expect you to be able to show, on an exam, how the existence of a PRG implies the existence of a one-way function). Showing that a one-way *permutation* implies a PRG follows the outline of what we showed in class, although the construction required a difficult theorem by Goldreich and Levin about the existence of hard-core bits. Showing that a one-way *function* suffices to construct a PRG is much more difficult, and we do not discuss it here.

4 Private-Key Encryption, Revisited

PRGs solve completely the problem of private-key encryption *when only a single message is encrypted*; even then, the scheme we gave is secure only against a ciphertext only attack.

Since encryption uses the PRG to generate a string which is used as in the one-time pad scheme, the scheme is insecure if it is used more than once: if $C_1 = M_1 \oplus G(sk)$ and $C_2 = M_2 \oplus G(sk)$ are two ciphertexts that are intercepted by an adversary, the adversary learns that $M_1 \oplus M_2 = C_1 \oplus C_2$. And, of course, the scheme is completely insecure against a known plaintext attack: if the adversary knows that C_1 is an encryption of M_1, the adversary learns $G(sk)$ (note that the adversary *does not* learn sk; but this is not needed to break the scheme!) and can then decrypt any subsequent ciphertexts that are sent. In practice, we would like users to be able to encrypt multiple messages; since these messages might be known (or even chosen!) by an adversary, we would also like an encryption scheme that is secure against chosen plaintext attacks.

Toward this goal, we are going to eventually have to rigorously define a notion of security against chosen plaintext attack (it turns out that this implies security when multiple messages are encrypted). If we remember our definition of (computational) security against ciphertext-only attacks, we had: for all PPT adversaries A and all messages m_1, m_2, the quantity

$$\left|\Pr[sk \leftarrow \{0,1\}^k; C_1 = \mathcal{E}_{sk}(m_1) : A(C_1) = 1] - \Pr[sk \leftarrow \{0,1\}^k; C_2 = \mathcal{E}_{sk}(m_2) : A(C_2) = 1]\right|$$

should be negligible. It seems kind of messy to modify this definition to handle chosen plaintext attacks. So, what we first need to do is to modify this definition so that it has a form more convenient to work with and to extend.

To do this, we define the notion of an *oracle*. An oracle is something that an adversary interacts with; when the adversary sends some input (a *query*) to the oracle, the oracle replies with some output (an *answer*, or *response*). The oracle is a "black-box" as far as the adversary is concerned: the adversary does not see how the outputs of the oracle are produced, and cannot tamper in any way with the functioning of the oracle. The adversary's view of this oracle is restricted to just whatever queries the adversary asks and the corresponding answers it receives. (Note that we frequently do not need to actually implement the oracle; it is simply an abstraction that we use in our definitions.)

Oracles can have any functionality one likes — that's part of what makes them so useful! The oracle we will be interested in for now is a *left-or-right encryption oracle* denoted $\mathsf{LR}_{b,sk}(\cdot,\cdot)$, whose exact specification depends on the encryption scheme of interest. The oracle is indexed by a bit b and a key sk. The oracle takes two inputs m_0, m_1 and returns the output $\mathcal{E}_{sk}(m_b)$; if the encryption scheme is randomized, then oracle chooses new random bits each time it responds to an input query. The name "left-or-right" comes from the fact that if $b = 0$ then the oracle always returns an encryption of the left message, and if $b = 1$ then the oracle always returns an encryption of the right message.

Our first notion of security (which we formalize rigorously in a moment) considers the following game based on any encryption scheme: a random key sk and a random bit b are chosen. This defines oracle $\mathsf{LR}_{b,sk}(\cdot,\cdot)$ (the encryption scheme is implicit here). The adversary can now submit a *single* query (m_0, m_1) to this oracle, which returns answer $\mathcal{E}_{sk}(m_b)$. Given this, the adversary now tries to predict the value of b that was used by the oracle. We say that the encryption scheme is secure if the adversary's probability of correctly guessing b is not better than $1/2$ by more than a negligible quantity.

More formally, given any encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$ (secret keys are assumed to be chosen uniformly at random from $\{0,1\}^k$) and any adversary A, define the *success probability* *of A in attacking* Π as:

$$\mathsf{Succ}_{A,\Pi}(k) \stackrel{\text{def}}{=} \Pr[sk \leftarrow \{0,1\}^k; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,sk}(\cdot,\cdot)}(1^k) = b],$$

where the adversary is allowed to query the LR oracle only *once*. Again, this is just the probability that the adversary correctly guesses b. (Note the notation here: $A^{\mathsf{LR}_{b,sk}(\cdot,\cdot)}$ denotes an adversary given oracle access to $\mathsf{LR}_{b,sk}(\cdot,\cdot)$.) Finally, we say the scheme is secure if the following is negligible:

$$|\mathsf{Succ}_{A,\Pi}(k) - 1/2|\,.$$

You should convince yourself that this definition is exactly equivalent to our old definition of computational security.

4.1 Extending the Definition

A nice feature of this new definition is that is easily extends to model security against chosen plaintext attacks. In the definition thus far, we restricted the adversary to making only a *single* query to the LR oracle; this was meant to model ciphertext only attacks. But we can strengthen the definition by allowing the adversary to query the oracle *polynomially-many* times (i.e., as many times as the polynomially-bounded adversary likes). In fact, for completeness we present this definition in its entirety, and call the new notion *security in the sense of indistinguishability* (and call a scheme satisfying the notion *indistinguishable*).

Definition 1 *An encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$ is* secure in the sense of left-or-right indistinguishability *(i.e., secure against chosen plaintext attacks) if, for all* PPT *adversaries A the following is negligible:*

$$\left|\Pr[sk \leftarrow \{0,1\}^k; b \leftarrow \{0,1\} : A^{\mathsf{LR}_{b,sk}(\cdot,\cdot)}(1^k) = b] - 1/2\right|.$$

(We no longer restrict the number of times the adversary may query LR*; of course, since A is a* PPT *algorithm it can query* LR *at most polynomially-many times.)*

You should convince yourself that this definition, in particular, implies security even when multiple messages are encrypted, and also implies security against chosen plaintext attacks and (of course) known plaintext attacks.

You should also be able to see that *no deterministic (stateless) encryption scheme can satisfy this definition* (why?). So our next challenge will be to consider randomized encryption schemes. It also seems that PRGs are not quite enough to obtain an encryption scheme satisfying this definition; in the next lecture we will introduce some more powerful machinery that will enable a solution.