1 Last lecture, and plan for today's lecture

Last lecture, we showed that the existence of PRPs implies the existence of PRFs, and vice versa. We also introduced the idea of statistical distance, and used it to show that a random function looks like a random permutation. The key points to keep in mind w.r.t statistical distance:

- if statistical distance of two distributions is low, then the distributions are indistinguishable.
- statistical distance satisfies the triangle inequality. Hence, we can say that if $\mathcal{D}_1$ is close to $\mathcal{D}_2$ (via statistical distance), and $\mathcal{D}_2$ is close to $\mathcal{D}_3$ (via statistical distance), then $\mathcal{D}_1$ is close to $\mathcal{D}_3$.

This lecture, we will conclude our discussion on PRFs/PRPs, and start discussing stronger definitions of encryption security.

2 The One-Way-Function Tree

Having completed our discussion of PRPs vs PRFs, this is a good time to recall the 'one-way-function-tree'. A lot of interesting primitives can be built from OWFs. It is somewhat surprising that even highly structured objects like PRPs can also be built from OWFs. This is summarized pictorially below (Figure 1). One exotic primitive in this picture is a *one-way permutation*. Currently, we don't know how to construct OWPs from OWFs, and we have good reasons to believe that OWPs **cannot** be built from OWFs.

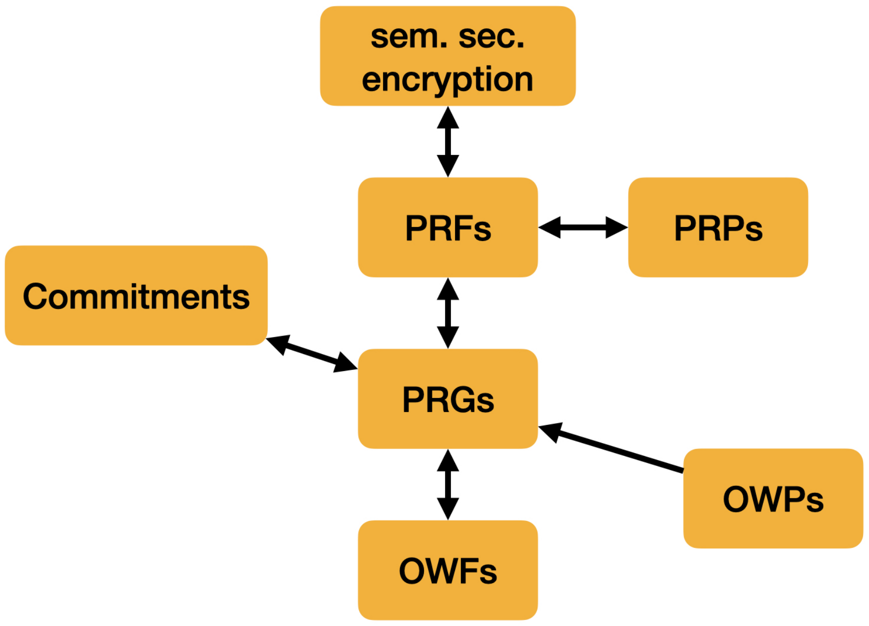

Figure 1: Primitives that can be built from OWFs. We did not discuss how to build PRGs from OWFs or OWPs. I will try to include the OWP $\rightarrow$ PRG construction in one of the future assignments.

2.1 Extending the co-domain of a PRF

So far, in most of our PRF related discussions, we have conveniently set the key space, input space and the output space to be $\{0,1\}^n$. However, this is not inherent to the definition of PRFs/PRPs. There can be certain situations where we need PRFs with larger output space.

Suppose we are given a PRF F with key space $\{0,1\}^n$, input space $\{0,1\}^n$, output space $\{0,1\}^n$. Can we construct a PRF with key space $\{0,1\}^n$, input space $\{0,1\}^n$ and output space $\{0,1\}^{2n}$?

Besides being a natural question, this exercise will also help us understand PRF security better. In the following section, I've listed the candidates that were proposed in class. Some of them were broken, and some may/may not be secure. Understanding these 'broken/plausibly broken' constructions is also an important exercise, since it tells us how to construct provably secure constructions.

2.1.1 Attempts discussed in class

1. $F'(x,k) = F(x,k) \mid\mid F(x \oplus 0^{n-1}1, k)$: This fails for the same reason as the previous attempt. First the adversary queries on some arbitrary x, receives $(y_1||y_2)$ then it queries on $x \oplus 0^{n-1}1$ and receives $z_1||z_2$. It checks if $y_2 = z_1$, and by same reasoning as above, it can distinguish $F'(\cdot,k)$ from a random function.

2. $F'(x,k) = F(x,k) \mid\mid F(F(x,k),k)$: The function F' will not be a secure PRF. Consider the following adversary that queries on some arbitrary string, say 0^n. It receives $y_1||y_2$ in response. Next, it queries on y_1 and receives $z_1||z_2$. Now, note that if the challenger was using $F'(\cdot,k)$, then z_1 would be equal to y_2. If the challenger was using a random function, then with high probability, $y_2 \neq z_1$. Therefore, these two queries help to distinguish $F'(\cdot,k)$ from a random function.

3. $F'(x,k) = F(0^{n/2}||x_{[n/2]}, k) \mid\mid F(1^{n/2}||x_{[n/2+1,n]}, k)$: This is broken because the adversary can query on two strings that have the same first $n/2$ bits, and in both responses, the first n bits will be the same.

4. $F'(x,k) = F(x,k) \mid\mid F(x, k \oplus 0^{n-1}1)$: Here, we are using related keys (k and $k \oplus 0^{n-1}1$). PRF security only guarantees security if either same key is used for all queries, or **unrelated** keys are used (in this case, we will prove security via a hybrid argument). You should be able to construct secure PRFs that are not related-key secure (that is, the resulting function F' will not be a secure PRF).

5. $F'(x,k) = F(x,k) \mid\mid F(k,x)$: This construction is not immediately broken, but if you try to prove security (assuming PRF security of F), then you should get stuck.

   Intuitive reason why the reduction doesn't work: PRF security only guarantees indistinguishability for a random key. Here, the second component uses the adversarially chosen input x as a key. Even in a secure PRF, there can be certain keys that are 'bad keys', and leak the complete input.

   There exist secure PRFs where the above transformation will result in an insecure PRF F'. Consider a function F which is a secure PRF, but $F(x, 0^n) = x$ (that is, for a random key, it is indistinguishable from a uniformly random function, but if the key is 0^n, then the function reveals the input). If we use this F for constructing F', then $F'(0^n,k) = F(0^n,k) \mid\mid k$, and as a result, the adversary learns k.

6. $F'(x,k) = F(x,k) \mid\mid F(x, F(x,k))$: This construction is also broken, because we know the key that is used for the second component. As a result, given output (y_1,y_2), the adversary can check if $F(x,y_1) = y_2$.

7. $F'(x,k) = F(x,k) \mid\mid F(k, F(x,k))$: This construction is somewhat tricky to analyse (note that the previous attack does not work anymore, since the adversary does not know k). However, the issue here is that the adversary has some control over the key being used in the second component, and this can be dangerous.

   The intuition is to design F such that $F(k, F(0^n,k))$ is some fixed string, say 0^n. Consider the set S of all pairs $(k, F(0^n,k))$. There are 2^n such pairs, which is a very sparse set within $\{0,1\}^n \times \{0,1\}^n$. As a result, if we take a secure PRF, and 'alter' the behaviour on this set S, this change is not noticed for a random PRF key.

   A PRF F that results in an insecure F': Let $\tilde{F}$ be a secure PRF. We will use $\tilde{F}$ to construct a keyed function F that is a secure PRF, but the above transformation will result in an insecure PRF.

$$F(w,z) = \begin{cases} 0^n \text{ if } \tilde{F}(0^n, w) = z \\ \tilde{F}(w,z) \text{ otherwise} \end{cases}$$

This function F is a secure PRF, assuming $\tilde{F}$ is (I am not including the proof here since it is a bit complicated and might be a distraction in the notes. If you are interested in discussing this, I am happy to discuss this proof in-person). Also, note that if we build F' using F, then F', when queried on 0^n, will have 0^n in its last n bits.

Summary of the approaches discussed: In each of the approaches above, we used the function F twice. Let x be the input, and k the key for function F'. The approaches discussed above can be broadly classified into the following categories:

- the key is same for both applications of F, but the inputs are either directly related to each other (eg. attempt 1), or they're indirectly related (eg. attempt 2). In such settings, the adversary can make two queries to F', and the outputs will be correlated.
- the keys are different, but both keys are derived from k : this can lead to related keys attacks (eg. attempt 4).
- the keys are different, but the adversary has some control on the keys used for one/both the applications of F: this can happen if the input x is directly/indirectly used as a key for one of the F applications. This can again lead to an insecure PRF F' (eg attempts 5, 6).

2.1.2 A construction that works

The following construction (also proposed in class) is a provably secure construction. First, note that a PRG $G : \{0,1\}^n \to \{0,1\}^{2n}$ can be constructed from a PRF F. Therefore, we will describe F' in terms of a pseudorandom function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and a pseudorandom generator $G : \{0,1\}^n \to \{0,1\}^{2n}$.

$$F'(x,k) : \text{compute } G(k) = (k_1, k_2)$$
$$\text{Output } F(x,k_1) \mid\mid F(x,k_2)$$

Intuitively, this is secure, because using (k_1, k_2) is 'almost-like' using two independent PRF keys. The proof is via a sequence of hybrid experiments. First, world-0 is where the challenger picks a random PRF key k, and uses $F'(\cdot, k)$ for responding to key queries.

<u>World-0</u>

- Challenger chooses PRF key $k \leftarrow \{0,1\}^n$. It computes $G(k) = (k_1, k_2)$.
- For each query x, the challenger sends $F(x,k_1) \mid\mid F(x,k_2)$.
- Adversary sends its guess b' after polynomially many queries.

<u>Hybrid-1</u>

- Challenger chooses $\underline{k_1 \text{ and } k_2 \text{ uniformly at random from } \{0,1\}^n}$.
- For each query x, the challenger sends $F(x,k_1) \mid\mid F(x,k_2)$.
- Adversary sends its guess b' after polynomially many queries.

<u>Hybrid-2</u>

- Challenger chooses $\underline{\text{uniformly random function } f_1 \leftarrow \mathsf{Func}[\mathcal{X},\mathcal{X}]}$ and k_2 uniformly at random from $\{0,1\}^n$.
- For each query x, the challenger sends $\underline{f_1(x)} \mid\mid F(x,k_2)$.

- Adversary sends its guess b' after polynomially many queries.

Hybrid-3

- Challenger chooses $\underline{\text{uniformly random functions } f_1, f_2 \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{X}]}$.
- For each query x, the challenger sends $f_1(x) \mid\mid \underline{f_2(x)}$.
- Adversary sends its guess b' after polynomially many queries.

World-1

- Challenger chooses $\underline{\text{uniformly random function } f \leftarrow \mathsf{Func}[\mathcal{X}, \mathcal{Y}] \text{ where } \mathcal{Y} = \{0,1\}^{2n}}$.
- For each query x, the challenger sends $\underline{f(x)}$.
- Adversary sends its guess b' after polynomially many queries.

Indistinguishability of hybrids: World-0 and Hybrid-1 are indistinguishable due to PRG security. Hybrid-1 and Hybrid-2 are indistinguishable due to PRF security. Similarly, Hybrid-2 and Hybrid-3 are indistinguishable due to PRF security. Finally, note that Hybrid-3 and World-1 are identical (choosing a random function mapping n bits to $2n$ bits is same as choosing two random function mapping n bits to n bits, and concatenating their outputs).

Note that this is just a proof sketch; for a formal proof, you must give reductions for each of these steps.

2.2 Other implications of PRFs

PRFs/PRPs also give us a natural class of functions that should be 'hard to learn'. Think of a PRF with input space $\{0,1\}^n$, key space $\{0,1\}^n$ and output space $\{0,1\}$. For a randomly chosen key k, the function $F(\cdot, k)$ can be seen as a classifier that labels each input with a 'yes' label or a 'no' label (depending on the output of $F(\cdot, k)$). Assuming F is a secure PRF, this labeling is hard to learn.

Qn: Can we build PRGs from PRPs?
Ans: Yes, as shown in the lecture, any PRP is also a PRF. And a PRF can be used to build a PRG (as discussed in one of the previous lectures).

3 Encryption schemes secure against all 'read-only' adversaries: Security definition

In this section, we will first define security against all 'read-only' adversaries, and then show a construction based on PRFs. Consider a real-world scenario where Alice and Bob are exchanging ciphertexts over an insecure channel, and the adversary can read all the ciphertexts. Earlier, when we discussed $\mathsf{No\text{-}Query\text{-}Semantic\text{-}Security}$, the threat model assumed that the adversary sees exactly one ciphertext, which corresponds to either m_0 or m_1. But in this general scenario, the adversary can see many ciphertexts. Suppose the adversary knows that the i^{th} ciphertext corresponds to either $m_{i,0}$ or $m_{i,1}$. Maybe the adversary knows the underlying messages for some of the ciphertexts, and wants to figure out the underlying messages for the other ciphertexts. We want the strongest definition for this 'read-only' model. The following are some of the security games proposed in class.

3.1 Security games proposed in class

- In the first attempt, the challenger picks a new key for each challenge pair. This security definition does not capture the scenario where the same key is used to encrypt many messages. In fact, you should show that any scheme that satisfies No-Query-Semantic-Security also satisfies this definition (the security game is defined in Figure 2.

Semantic Security - Attempt 1

1. **Setup:** Challenger chooses a bit $b \leftarrow \{0, 1\}$.
2. **Challenge encryption queries:** Adversary sends polynomially many challenge encryption queries (adaptively). The i^{th} challenge pair consists of two messages $m_{i,0}, m_{i,1}$.

   For each query, the challenger chooses a uniformly random key k_i, and sends $\mathsf{ct}_i = \mathsf{Enc}(m_{i,b}, k_i)$ to the adversary.

   Note that the bit b was chosen during setup, and the same bit is used for all queries.
3. **Guess:** The adversary sends its guess b', and wins the security game if $b = b'$.

Figure 2: Semantic Security Game - Attempt 1

Question 14.01. Show that if an encryption scheme $\mathcal{E}$ satisfies No-Query-Semantic-Security, then it also satisfies security defined via the first attempt (shown in Figure 2).

- In the second proposal, the attacker queries for a few ciphertexts, and must output a new message along with a ciphertext for the new message. It is defined in Figure 3.

  This definition does not capture the intuition that the ciphertext hides the message. For instance, consider a silly encryption scheme where the encryption of a message m using key k outputs m, along with a complicated function of (m, k). Clearly, it does not hide the message, but it is possible that no p.p.t. adversary is able to win the security game. As a result, this security game does not capture our threat model.

Semantic Security - Attempt 2

1. **Setup:** Challenger chooses key k.
2. **Encryption queries:** Adversary sends polynomially many challenge encryption queries (adaptively). The i^{th} query consists of message m_i.

   For each query, the challenger sends $\mathsf{ct}_i = \mathsf{Enc}(m_i, k)$ to the adversary.

   Note that the key k was chosen during setup.
3. **Guess:** The adversary sends a new message m' (that was not queried before) and ct'. It wins if $\mathsf{Dec}(\mathsf{ct}', k) = m'$.

Figure 3: Semantic Security Game - Attempt 2

- In the third security game proposed in class, the challenger chooses a key k, and uses the same key for either encrypting $m_{i,0}$ or $m_{i,1}$. The adversary gets to send polynomially many queries, and at the end, it must send its guess. The game is defined in Figure 4.

  Given this security game, the first thing to observe is that for all the encryption schemes that we've seen so far, there exists an adversary that wins with probability 1. The adversary simply queries for encryption of message m_0 and receives ct. Next, it sends two challenge messages m_0 and m_1. It receives ct^*. If $\mathsf{ct} = \mathsf{ct}^*$, then it guesses that m_0 was encrypted, else it guesses that m_1 was encrypted.

  But is that an issue with the security definition? Or should we somehow update our encryption scheme constructions so that they are no longer deterministic? If the encryption scheme is randomized, how do we ensure correctness? These are some of the issues that we will address in our next lecture.

Semantic Security

1. **Setup:** Challenger chooses an encryption key $k \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0, 1\}$.
2. **Challenge encryption queries:** Adversary sends polynomially many challenge encryption queries (adaptively). The i^{th} challenge pair consists of two messages $m_{i,0}, m_{i,1}$. The challenger sends $\mathsf{ct}_i = \mathsf{Enc}(m_{i,b}, k)$ to the adversary.

   Note that the bit b and key k were chosen during setup, and the same bit and key are used for all queries.
3. **Guess:** The adversary sends its guess b', and wins the security game if $b = b'$.

Figure 4: Semantic Security Game

4 Lecture summary, plan for next lecture, additional resources

Summary: Here are the main take-aways from this lecture:

- We saw a few attempts for extending the output space of a pseudorandom function. One of the attempts worked (proof sketch given in Section 2.1.2). Here are a few reasons why some of the other attempts failed:
  - part of the output depended on only part of the input.
  - part of the output on an input x revealed part of the output on a related input x'.
  - the different parts of the output were derived from related keys
  - part of the output was derived using the adversarially chosen input as PRF key, or some function of the adversarially chosen input as PRF key.
- Next, we saw a few attempts for defining semantic security (that is, security against all 'read-only' adversaries).

  Do not take the definitions on faith. You should question why a certain is a reasonable definition, and you are strongly encouraged to come up with your own security games.

Next Lecture: We will see a construction that satisfies semantic security.