COS 533: Advanced Cryptography — Princeton University
Lecture 2 (September 18, 2017)
Lecturer: Mark Zhandry — Scribe: Mark Zhandry

Notes for Lecture 2

1 Last Time

Last time, we defined formally what an encryption scheme is. A (symmetric key or secret key) encryption scheme consists of two algorithms $(\mathsf{Enc}, \mathsf{Dec})$. Enc is a PPT algorithm that takes as input a key and a plaintext, and outputs a ciphertext. Dec is deterministic polynomial time, takes as input a key and a ciphertext, and outputs a plaintext. For correctness, we require that when used with the same key, Dec inverts Enc. More precisely, for all messages m,

$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m, k \stackrel{\$}{\leftarrow} \{0,1\}^\lambda] = 1$$

Here, the probability is taken over a random k, and any random coins chosen by Enc.

For security, let A be an adversary. Let $\mathsf{IND\text{-}CPA\text{-}EXP}_b(A, \lambda)$ be the following experiment on A, parameterized by a bit b:

1. A interacts with a challenger, denoted Ch.
2. At first, Ch chooses a random key $k \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$
3. Next, A sends the challenger two messages m_0, m_1. Ch selects and encrypts m_b: $c \leftarrow \mathsf{Enc}(k, m_b)$. Then Ch sends c back to A.
4. A can repeat step 3 as many times as it wishes. We will charge A one unit of time for every time it repeats step 3.
5. Finally, A outputs a guess b' for b. b' is the output of $\mathsf{IND\text{-}CPA\text{-}EXP}_b(A, \lambda)$

Here, IND refers to indistinguishability, meaning that the adversary is trying to distinguish between two experiments, $b = 0$ and $b = 1$. CPA stands for "chosen plaintext attack". This refers to the fact that the adversary is able to choose the plaintexts that get encrypted.

Definition 1 *An encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *is* $\mathsf{IND\text{-}CPA}$ secure *(in words,* indistinguishable under a chosen plaintext attack*) if, for all PPT adversaries A, there exists a negligible function ϵ such that*

$$|\ \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_1(A, \lambda)]\ | < \epsilon(\lambda)$$

We will often simply call such a scheme “CPA secure”. Intuitively, this definition means that any guess b' the adversary makes is more or less independent of the actual bit b, since the probabilities for any guess under the two experiments are extremely close.

2 This Time

Starting today, and for the next couple lectures, we will show how to construct encryption and other cryptographic applications from weaker tools. In particular, we will show:

1. PRFs (pseudorandom functions) $\rightarrow$ CPA-secure secret key encryption
2. PRGs (pseudorandom generators) $\rightarrow$ PRFs
3. OWPs (one-way permutations) $\rightarrow$ PRGs
4. PRFs $\rightarrow$ MACs (message authentication codes)
5. MACs + CPA-secure secret key encryption $\rightarrow$ CCA-secure secret key encryption
6. PRFs + UOWHFs (universal one-way hash functions) $\rightarrow$ digial signatures (aka public key MACs)

Additionally, it is known how to improve step 3 to “OWF (one-way functions) $\rightarrow$ PRGs” and to build show that “OWF $\rightarrow$ UOWHFs”. Therefore, all of the crypto primitives above can be build from one-way functions.

Today, we will show the first two steps, namely how to build encryption from PRFs and PRFs from PRGs.

3 PRFs

A PRF is a keyed function that looks like a random function if you never get to see the key. That is PRF is a deterministic polynomial time computable function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{n(\lambda)} \rightarrow \{0,1\}^{m(\lambda)}$, with the following security property.

Let A be an adversary. Let $\mathsf{PRF\text{-}EXP}_b(A, \lambda)$ be the following experiment on A, parameterized by a bit b:

1. A interacts with a challenger, denoted Ch.

2. At first, if $b = 0$, Ch chooses a random key $k \xleftarrow{\$} \{0,1\}^\lambda$. If $b = 1$, Ch initializes an empty list L.

3. Next, A sends the challenger an input $x \in \{0,1\}^{n(\lambda)}$. Ch responds as follows

   - If $b = 0$, Ch responds with $y \leftarrow \mathsf{PRF}(k, x)$.
   - If $b = 1$, Ch looks for a pair (x, y) in L. If it finds an (x, y), it responds with y. Otherwise, it generates a random y and adds the pair (x, y) to L. Then it responds with y.

4. A can repeat step 3 as many times as it wishes. We will charge A one unit of time for every time it repeats step 3.

5. Finally, A outputs a guess b' for b. b' is the output of $\mathsf{PRF\text{-}EXP}_b(A, \lambda)$

Notice that in the $b = 1$ case, Ch is effectively providing A with a truly random function O where all outputs are chosen independently and uniformly at random. In the $b = 0$ case, Ch is providing A with the PRF on a random key k. A's goal is to distinguish the two cases.

Definition 2 *An PRF* PRF secure *if, for all PPT adversaries A, there exists a negligible function ϵ such that*

$$| \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_1(A, \lambda)] | < \epsilon(\lambda)$$

4 CPA-secure secret key encryption from PRGs

Let PRF be a pseudorandom function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{n(\lambda)} \rightarrow \{0,1\}^{m(\lambda)}$.

Our scheme will be the following:

- $\mathsf{Enc}(k, m)$ for $k \in \{0,1\}^\lambda$ and $m \in \{0,1\}^{m(\lambda)}$ does the following. First it chooses a random $r \in \{0,1\}^\lambda$, and computes $c \leftarrow \mathsf{PRF}(k, r) \oplus m$. It outputs (r, c)

- $\mathsf{Dec}(k, (r, c))$ computes $m \leftarrow \mathsf{PRF}(k, r) \oplus c$

Correctness is straightforward, since Dec computes $\mathsf{PRF}(k, r) \oplus c = \mathsf{PRF}(k, r) \oplus (\mathsf{PRF}(k, r) \oplus m) = m$.

Theorem 3 *If* PRF *is a secure PRF and $2^{n(\lambda)}$ is super polynomial, then* $(\mathsf{Enc}, \mathsf{Dec})$ *is CPA secure.*

Proofs in cryptography are usually proofs by contradiction: we assume an adversary violating the security of one primitive (in our case, an encryption scheme), and derive from it an adversary for some starting primitive (in our case, a PRF).

We will also introduce one of the standard proof techniques in cryptography, a hybrid argument. Assume toward contradition that there is an adversary A and a non-negligible function ϵ such that

$$| \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_1(A, \lambda)] | \geq \epsilon(\lambda)$$

We will define several "hybrid" games, where the first is $\mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)$, and the last is $\mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)$. By our assumption, we know that A distinguishes the first and last hybrid. Therefore, it must also distinguish some pair of adjacent intermediate hybrids. We will use such a distinguishing advantage to break the security of the PRF.

- **Hybrid 0** is identical to $\mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)$. Substituting in to the experiment our construction, the experiment works as follows:

    1. At first, Ch chooses a random key $k \xleftarrow{\$} \{0,1\}^\lambda$
    2. Next, A sends the challenger two messages m_0, m_1. Ch chooses a random r in $\{0,1\}^{n(\lambda)}$, and computes $y \leftarrow \mathsf{PRF}(k, r)$. Then Ch sends $(r, y \oplus m_0)$ back to A. A can repeat this step as many times as it wishes. s

- **Hybrid 1** is the following modifications to the $\mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)$ experiment.

    - Instead of generating a random key k, Ch initializes an empty list L.
    - Instead of computing $y \leftarrow \mathsf{PRF}(k, r)$, Ch looks for (r, y) in L, using y if found. Otherwise, it generates a fresh random y, and adds (r, y) to L.

- **Hybrid 2** is the same as **Hybrid 1**, except that Ch sends $(r, y \oplus m_1)$ back to A.

- **Hybrid 3** is the same as **Hybrid 2**, except that Ch goes back to choosing a random key k and setting $y \leftarrow \mathsf{PRF}(k, r)$. Notice that **Hybrid 3** is identical to $\mathsf{IND\text{-}CPA\text{-}EXP}_1(A, \lambda)$

Now, by our assumption that $(\mathsf{Enc}, \mathsf{Dec})$ is insecure and the triangle inequality, we have that there must exist an $i \in \{0, 1, 2\}$ such that

$$| \Pr[1 \leftarrow \text{Hybrid}_i(A, \lambda)] - \Pr[1 \leftarrow \text{Hybrid}_{i+1}(A, \lambda)] | \geq \frac{1}{3}\epsilon(\lambda)$$

We now consider the three cases:

- $i = 0$. Notice that the only difference between **Hybrid0** and **Hybrid1** is that in **Hybrid 0**, y is set to $\mathsf{PRF}(k, r)$, whereas in **Hybrid 1**, y is set to random. It is straightforward to construct a PRF adversary B which distinguishes PRF from random with advantage $\epsilon(\lambda)/3$. B works as follows: it simulates A, playing the role of CPA-security challenger to A. Whenever A makes a query (m_0, m_1), B chooses a random r, and queries its own PRF challenger on r, obtaining y. Then it responds to A with $(r, y \oplus m_0)$. Finally, when A outputs a bit b', B outputs b'.

  In $\mathsf{PRF\text{-}EXP}_0(A, \lambda)]$, B successfully simulates the view of A in **Hybrid 0**. Similarly, in $\mathsf{PRF\text{-}EXP}_1(A, \lambda)]$, B successfully simulates the view of A in **Hybrid 1**. Therefore, B's advantage is the same as A's in distinguishing these two hybrids, namely $\epsilon(\lambda)/3$. This is non-negligible, a contradiction to the assumed security of PRF.

- $i = 1$. In **Hybrid 1** and **Hybrid 2**, y is set to random; the only difference is that it is XORed with m_0 or m_1 before responding to A. However, a random string XORed with anything is still random, so A essentially receives random strings in both hybrids.

  The only potential problem if the same r is used to encrypt in two different queries. In this case, the response to each query is random, but the two responses are correlated.

  Such collisions in r are, however, not a common occurrence: the probability any two queries collide is $2^{-}n(\lambda)$. The probability that some pair of queries collide is therefore at most $q^2 \times 2^{-n(\lambda)}$ where q is the number of queries. Recall that q is a polynomial, and that $2^{-n(\lambda)}$ is negligible. Since a negligible function times a polynomial is still negligible, we have that the probability of a collision is negligible. Therefore, A's distinguishing advantage $\epsilon(\lambda)/3$ must be negligible, a contradiction.

- $i = 2$. This is handled identically to $i = 0$, except that B encrypts m_1 instead of m_0.

Therefore, in any of the three cases, we reach a contradiction. Therefore, our assumed adversary A could not possibly exist. This completes the proof of security.

5 PRGs

Next, we turn to constructing PRFs from a weaker object called a pseudorandom generator, or PRG. A PRG is a deterministic polynomial time function $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+s(\lambda)}$ for some $s(\lambda) > 1$. This means that G expands its input. For security, we ask that outputs of G look as if they were random.

Definition 4 *A function G is a secure PRG if, for all PPT adversaries A, there exists a negligible function ϵ such that*

$$| \Pr[A(G(x)) = 1 : x \xleftarrow{\$} \{0,1\}^{\lambda}] - \Pr[A(y) = 1 : y \xleftarrow{\$} \{0,1\}^{\lambda+s(\lambda)}] | < \epsilon(\lambda)$$

Notice that G can only take on at most 2^{λ} outputs, smaller than the $2^{\lambda+s(\lambda)}$ points in the co-domain. Therefore, it is impossible for G's outputs to be truly random. Nonetheless, PRG security says that the outputs *look* random to any polynomial-time adversary.

6 PRFs from PRGs

We will not formally define the actual PRF algorithm, but instead will describe it in words. We will assume G is length-doubling, meaning $s(\lambda) = \lambda$. First, let's forget efficiency for the moment. The PRF PRF works as follows. It takes the key $k \in \{0,1\}^{\lambda}$, and applies G. The result is a 2λ-bit string. PRF splits the string in half into two λ-bit strings. Then it applies G separately to both halves, obtaining a 4λ-bit string. PRF splits this into 4 λ-bits strings, and applies G to each string. It continues in this way for $n(\lambda)$ steps, for any desired polynomial λ.

The result is a $2^{n(\lambda)} \times \lambda$-bit string. PRF will interpret this string as $2^{n(\lambda)}$ separate λ-bit strings. The output of PRF on input x with be the xth string in this list. Therefore, PRF has inputs of length $n(\lambda)$ and outputs of length λ.

It will be useful to think of the PRF computation as a tree: at the root is the PRF key k. The children of a node containing x are the first and second half of $G(x)$. The tree has $n+1$ levels and 2^n leaves. The leaves are the outputs of the PRF.

As described, PRF runs in time roughly $2^{n(\lambda)}$, which is exponential.

Question: How to compute each block locally in time polynomial in n, without computing the entire list of outputs?

Theorem 5 *If G is a secure PRG, then the construction above is a secure PRF*

As before, we will prove this theorem by a hybrid argument. Assume toward contradiction that there is an adversary A and a non-negligle function ϵ such that

$$| \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{PRF\text{-}EXP}_1(A, \lambda)] | \geq \epsilon(\lambda)$$

Define **Hybrid** i for $i \in \{0, \ldots, n\}$ as follows. **Hybrid 0** is just $\mathsf{PRF\text{-}EXP}_0(A, \lambda)$, where A interacts with PRF. In **Hybrid 1**, we slightly modify the experiment. Instead of choosing a random key $k \in \{0,1\}^\lambda$ and placing it at root of the PRF tree, Ch chooses two random strings x_0, x_1, and places them at level 1 of the tree (here we zero index the tree levels). It computes all nodes below the level 1 just as in the PRF: the level 2 is obtained by applying G to the elements of level 1, etc.

Hybrid 2 is defined analogously: choose random $x_{00}, x_{01}, x_{10}, x_{11}$, place them in level 2 of the tree, and generate the nodes levels 3 through n as before. Similarly define the remaining hybrids.

Note that in **Hybrid** n, all the leaves of the tree have uniformly random elements; this corresponds to A interacting with a truly random function, namely $\mathsf{PRF\text{-}EXP}_1(A, \lambda)$. Therefore, there exists an $i \in \{0, \ldots, n-1\}$ such that

$$\mid \Pr[1 \leftarrow \text{Hybrid}_i(A,\lambda)] - \Pr[1 \leftarrow \mathsf{Hybrid}_{i+1}(A,\lambda)] \mid \geq \frac{\epsilon(\lambda)}{n(\lambda)}$$

Notice that $\epsilon(\lambda)/n(\lambda)$ is non-negligible, since n is a polynomial. However, this still does not give us a PRG adversary. To finally get a PRG adversary, we introduce another sequence of hybrids **Hybrid** $i.j$ for $j \in \{0, \ldots, 2^i\}$. **Hybrid** $i.0$ is taken to be **Hybrid** i, namely we fill level i with uniformly random elements, and derive levels $i+1$ through n using G. In **Hybrid** $i.j$, we fill the first $2j$ elements of level $(i+1)$ with random elements, and the last $2^i - j$ elements of level i with random elements. For nodes $2j+1, \ldots, 2^(i+1)$ of level $(i+1)$ are derived from the parents using G. The rest of the nodes in levels $i+2$ up to n are derived as before using G. Notice that **Hybrid** $i, 2^i$ is identical to **Hybrid** $i+1$ since we are filling level $i+1$ with random elements. Therefore, similar to above, we find that there is a j such that

$$\mid \Pr[1 \leftarrow \text{Hybrid}_{i.j}(A,\lambda)] - \Pr[1 \leftarrow \mathsf{Hybrid}_{i.(j+1)}(A,\lambda)] \mid \geq \frac{\epsilon(\lambda)}{n(\lambda)2^i}$$

At this point, the views of A in these two hybrids differ only by a single node where $G(x)$ was replaced with random. Hence, it is straightforward to construct a PRG adversary B from A. Roughly, B, on input y, chooses random elements to fill the first nodes 1 through j of level i and the nodes $2j+3$ through 2^{i+1} of level $i+1$. For nodes $2j+1, 2j+2$ of level $i+1$, it puts in y. Then is simulates A with access to the tree derived from these nodes, and outputs the result of A. If $y = G(x)$ for a random x, then the view of A is identical to **Hybrid** $i.j$, and if y is random, then the view of A is identicalto **Hybrid** $i.(j+1)$. Therefore, B's distinguishing advantage is the same as the distinguishing advantage of A in these two hybrids, so we have that

$$\mid \Pr[1 \leftarrow B(G(x)) : x \xleftarrow{\$} \{0,1\}^\lambda] - \Pr[1 \leftarrow B(y) : y \xleftarrow{\$} \{0,1\}^{2\lambda}] \mid \geq \frac{\epsilon(\lambda)}{n(\lambda)2^i}$$

Unfortunately, i could be as large as n, and so the value on the right side could be exponentially small. Therefore, we do not necessarily get a contradiction. A more clever argument is therefore required.

The key insight is to remember that A is an efficient adversary, and can therefore only make a polynomial number $q(\lambda)$ of queries. In the PRF tree, this means that A only gets to see q of the leaves of the tree. If we trace these leaves up to level i, we see that the view of A only depends on at most q nodes in level i. For nodes j that A does not depend on, the hybrids **Hybrid** $i.j$ and **Hybrid** $i.(j+1)$ are actually perfectly indistinguishable, since A never even sees the nodes that depend on the change between the hybrids.

This means we can really skip all but q of the hybrids, obtaining an adversary B such that

$$\left|\ \Pr[1 \leftarrow B(G(x)) : x \xleftarrow{\$} \{0,1\}^\lambda] - \Pr[1 \leftarrow B(y) : y \xleftarrow{\$} \{0,1\}^{2\lambda}]\ \right| \geq \frac{\epsilon(\lambda)}{n(\lambda)q(\lambda)}$$

Note that the description above is not quite accurate, and basically assumed that A queries on a fixed set of nodes that do not depend on the results of previous queries. However, with a little care, it is possible to make the proof work even if A makes adaptive queries based on previous responses.

7 Extending the Length of PRGs

Above, we assume that G was length-doubling, in other words $s(\lambda) = \lambda$. Here, we briefly explain how to build such a G from one where $s(\lambda) = 1$.

The idea is similar to above, but we use a chain instead of a tree. Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$. Construct $G' : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ as follows. On input x, run G, and write its output as (x_1, b_1) for $x_1 \in \{0,1\}^\lambda$ and $b_1 \in \{0,1\}$. Then apply G again, this time to x_1, obtaining x_2, b_2. Repeat this process λ times until you have $x_\lambda, b_1, \ldots, b_\lambda$. Output these as the output of G. Security can be proved through a similar (but simpler) proof as above.