Lecture 5 - CPA security, Pseudorandom functions

Boaz Barak

October 2, 2007

Reading Pages 82–93 and 221–225 of KL (sections 3.5, 3.6.1, 3.6.2 and 6.5). See also Goldreich (Vol I) for proof of PRF construction.

Quick review The PRG Axiom, encryption schemes with key size $\ll$ message size.

Stronger encryption schemes. Last time we showed that get a *single message* encryption scheme with key arbitrarily smaller than the message. However, in real life we want to encrypt *multiple* messages. When thinking about multiple messages security, we need to consider the question of where do these messages come from. We can't be sure that they are not affected in some way by our adversary (remember the Brits & Enigma in WWII). Therefore, we want security against *chosen plaintext attack*.

CPA Secure Encryption scheme. This is the following game:

- Adversary chooses x_1, x_2.
- Sender chooses $k \leftarrow_R \{0,1\}^n$, $i \leftarrow_R \{1,2\}$ and sends $y = \mathsf{E}_k(x_i)$ to the adversary.
- For as long as adversary desires (but less than poly(n)— its running time), adversary chooses a message x and gets $\mathsf{E}_k(x)$. Note that it is legitimate for the adversary to choose $x = x_1$ or $x = x_2$ but it can also choose other messages. (Notation: adversary has *oracle* access to the mapping $x \mapsto \mathsf{E}_k(x)$.)
- Adversary comes up with a guess j. She *wins* if $i = j$.

(E, D) is CPA-secure if for every poly Adv, poly-bounded $\epsilon : \mathbb{N} \to [0,1]$, and large enough n, $\Pr[\mathsf{Adv} \text{ wins }] \leq 1/2 + \epsilon(n)$.

Note: a deterministic scheme can't be CPA secure (see also exercise).

Constructing a CPA secure scheme. It is not immediate how to construct such a scheme from a pseudorandom generator. To do that, we'll use a new creature called *pseudorandom functions* (PRF). PRFs have many other applications in cryptography and seem quite amazing, but they can be constructed based on any pseudorandom generator.

Pseudorandom functions A random function $F(\cdot)$ from n bits to n bits can be thought of as the following process: for each one of its possible 2^n inputs x, choose a random n-bit string to be $F(x)$. This means that we need $2^n \cdot n$ coins (which is *a lot*) to choose a random function. We also need about that much size to store it.

We see that a function that can be described in n bits is very far from being a random function. Nevertheless we'll show that under our Axiom, there exists a *pseudorandom* function collection

that can be described and computed with poly(n) bits but is indistinguishable from a random function.

We let $\mathcal{F} = \{f_s\}_{s\in\{0,1\}^*}$ be a *collection of functions*. Suppose that $f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}$ (this is not important and we can generalize the definition and constructions to different input and output lengths). We say that the collection is *efficiently computable* if the mapping $s, x \mapsto f_s(x)$ is computable in polynomial time. Fix an efficiently computable collection and consider the following two games:

Game 1:

- s is chosen at random in $\{0,1\}^n$.
- Adversary gets black-box access to the function $f_s(\cdot)$ for as long as it wishes (but less than its poly(n) running time).
- Adversary outputs a bit $v \in \{0,1\}$.

Game 2:

- A random function $F : \{0,1\}^n \to \{0,1\}^n$ is chosen.
- Adversary gets black-box access to the function $F(\cdot)$ for as long as it wishes (but less than its poly(n) running time).
- Adversary outputs a bit $v \in \{0,1\}$.

$\mathcal{F}$ is a *pseudorandom function* (PRF) ensemble, if for every poly-time Adv and poly-bounded $\epsilon : \mathbb{N} \to [0,1]$, and large enough n

$$\Big|\Pr[\mathsf{Adv} \text{ outputs 1 in Game 1}] - \Pr[\mathsf{Adv} \text{ outputs 1 in Game 2}]\Big| < \epsilon(n)$$

GGM result Intuitively, it is not at all clear that such functions should exist. However, it was proven by Goldreich Goldwasser and Micali that if PRG exist then so do PRFs. (The other direction is pretty easy — can you see why?)

This means that under our "axiom" we have PRFs, so before describing this proof, let's see how we can use PRFs to get CPA-secure encryptions.

A CPA secure encryption . We construct the following encryption scheme:

- Key $k \leftarrow_{\mathrm{R}} \{0,1\}^n$.
- To encrypt a message $x \in \{0,1\}^n$, $\mathsf{E}_k(x)$ chooses r at random in $\{0,1\}^n$ and sends $\langle r, f_k(r) \oplus x\rangle$. Note that this is a **probabilistic encryption**.
- Given $\langle r, y\rangle$ to decrypt compute $f_k(r) \oplus y$.

Security

Theorem 1. (E, D) *is CPA secure.*

The below proof is somewhat sketchy, see KL for a full proof.

Proof. Let Adv be a poly-time adversary breaking (E, D) in a CPA attack with probability ϵ. We'll use this to break the security of the PRF.

The idea is to show that the scheme will be *statistically* secure (regardless of the running time of the adversary, as long as it makes less than $2^{n/10}$ queries) if we used *completely random* functions. Then, we'll say that if the scheme is not secure we can convert an adversary into a distinguisher for the PRF family.

Claim 2. *Let $(\mathsf{E}^I, \mathsf{D}^I)$ be the "imaginary" scheme where the parties share as a key a* random *function $F(\cdot)$.*[1] *Then, the probability that* Adv *guesses x_i in a CPA attack is less than $1/2 + 2^{-n/10}$.*

Proof. The adversary's guess is a function of the messages that it sees. Let T be the total number of queries it makes, and assume w.l.o.g that $T \ll 2^{n/10}$. Let's consider these messages that the adversary sees when $i = 1$ and $i = 2$. Denote the distribution of these messages by $Y_0^{(1)}, \ldots, Y_T^{(1)}$ and $Y_0^{(2)}, \ldots, Y_T^{(2)}$ where $Y_0^{(i)} = F_k(r) \oplus x_i$.

Let $r_1, \ldots, r_T$ be all the random strings chosen by the sender during the CPA attack. For a fixed i and j the probability that $r_i = r_j$ is 2^{-n}. Therefore by the union bound, the probability that there exists i and j such that $r_i = r_j$ is at most $T^2 2^{-n} < 2^{-n/2}$. This means that this event ($r_i = r_j$ for $i \neq j$) happens with so low probability we can ignore it and essentially assume it never happens.

Now if every r_i is different then no matter what the encrypted value is, all the messages $Y_T^{(i)}, \ldots, Y_T^{(i)}$ are independent and uniform. This is because we can think of the following "lazy" evaluation of the function F: we only choose F's output on a new value r when we are asked of it. However, if all the r's are distinct then every time the function is evaluated we choose a fresh random value. □

We use this result to transform a successful CPA adversary for (E, D) into a successful distinguisher for the PRF family. □

Note: It's a **very** good exercise to compare this proof sketch to the full proof in the KL book— this will show you how in general we transform a proof idea into an actual proof in cryptography.

Construction of PRFs As you can see on the web page, there are several candidate constructions for PRFs. However, for us the important thing is that we don't need to introduce a new axiom, since we can construct them directly from ordinary PRG. That is, we have the following theorem:

Theorem 3. *IF the PRG Axiom is true, then there exist pseudorandom functions.*

Proving Theorem 3 The best way to think about the construction is the following. Suppose that you have a PRG $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{2n}$. Construct a depth n full binary tree, which you label as follows: the root is labeled with a string s (the seed of the function). For each non-leaf node labeled v, the two children are labeled with $\mathsf{G}_0(v) \triangleq \mathsf{G}(v)_{[1\ldots n]}$ and $\mathsf{G}_1(v) \triangleq \mathsf{G}(V)_{[n+1\ldots 2n]}$.

We have 2^n leaves and we can identify each one of them with a string in $\{0,1\}^n$ in the natural way (the string depicts the path from the root to the leaf with 0 meaning take the left child

[1] Note that this imaginary scheme uses a key much longer than the total length of all messages.

and 1 meaning take the right child). We define $f_s(x)$ to be the label of the leaf corresponding to x.

Although the full tree is of exponential size to compute $f_s(x)$ we only need to follow an n-long path from the root to the leaf and so it is computable in polynomial time.

Another way to state this definition is that $f_s(x)$ is defined to be

$$\mathsf{G}_{x_n}(\mathsf{G}_{x_{n-1}}(\cdots \mathsf{G}_{x_1}(s))\cdots)$$

Analysis We now show the following result:

Lemma 4. *If $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{2n}$ is a pseudorandom generator, then the construction described above is a PRF collection.*

Proof. Suppose for the sake of contradiction that there is an T-time adversary Adv that manages to distinguish between access to $f_s(\cdot)$ and access to a random function with probability at least ϵ. We'll convert it to a T' adversary that manages to distinguish between $\mathsf{G}(U_n)$ and U_{2n} with probability at least ϵ', for T' and ϵ' polynomially related to T, ϵ.

Without loss of generality. We're going to make some modifications to the behavior of Adv which will not change its distinguishing probability and not add too much to its running time but will make our life a little easier. Since such modifications can be made, we can just *assume* that Adv is already of the modified form. That is, we assume the following about Adv:

- It makes exactly T queries: if it makes less, we'll change it to ask "meaningless" queries.
- It never asks the same question twice: we can modify Adv to keep track of all the responses it received from its oracle and whenever it wants to get an answer for a query it already asked, it can use that table.

We now consider the interaction of Adv with an oracle computing $f_s(\cdot)$. The algorithm we specified for f_s is a stateless algorithm that given s and x computes $f_s(x)$ without relying on any precomputed information. However, we can implement the oracle in any way we want as long as it still computes $f_s(\cdot)$. Thus, we'll implement it in the following way:

Description of the $f_s(\cdot)$ oracle. The oracle will build keep the binary tree we described above. Of course it cannot keep the entire tree, but it will build it and maintain it in response to each query of Adv.

- Initially the tree contains only the root which is labeled with s.
- Whenever Adv makes a query for $f_s(x)$, the oracle will look at the path from the leaf x to the root. Let v be the lowest point in the path which is already computed. The oracle will compute all the values along the path from v to x and store the labels, finally returning the label of x.

Note: Whenever the oracle invokes G on a label x of an internal (non-leaf) node v, it will label the children of v with $x_0 = \mathsf{G}_0(x)$ and $x_1 = \mathsf{G}_1(x)$ and *erase* the label of v. Note that this is OK since the oracle will never need to use these values again. Also note that the oracle needs to make at most $M = T \cdot n$ invocations of G during the entire process.

The hybrids. We are going to use a hybrid argument to prove that the interaction of Adv with this oracle is indistinguishable from an interaction with a random function. For $i = 0, \ldots, M$ we define the hybrid H^i in the following way:

This is the adversary's view in an interaction with the oracle *except* that for the first i times when the oracle is supposed to invoke G to label the two children of some node v labeled x, the oracle does *not* do this but rather does a "fake invocation": instead of labeling v's children with $(x_0, x_1) = \mathsf{G}(x)$ it chooses x_0, x_1 at random from $\{0,1\}^n$ and labels the two children with x_0, x_1, erasing the label of v.

Clearly H^0 is equal to the adversary's view when interacting with f_s while H^M is equal to the adversary's view when interacting with a random function.

Thus, we only need to prove that H^i is indistinguishable from H^{i-1}. However, this follows from the fact that G is a pseudorandom generator.

Proof of indistinguishability of H^i and H^{i-1}. We'll make the following modification to the operation of the oracle in H^i: in the first i "fake invocations" of G, when the oracle chooses at random x_1 and x_2 and uses these to label the nodes of v, it will do something a bit different: it will erase the label of v but use a "lazy" evaluation: it will mark the children of v as "to be chosen at random" and will choose each of these labels at random only when it will be needed at a future time. (Note that typically the label for one of the children will be needed in the next step, but the label for the other child may only be required to answer a future query or perhaps never). Even the root s is not chosen initially but rather is initiated with the "to be chosen at random" label. Note that for the first i "fake invocations" whenever the value for an internal node is used then it is immediately deleted, and so in the first i steps all the internal nodes are either untouched or marked "to be chosen at random". The important observation is that all this is only about the oracle's internal computation and has no effect on the view of the adversary. (Also, the oracle can stop being lazy and choose values for some of the nodes without any effect on the view.)

We'll now prove the indistinguishability. Suppose we had a distinguisher C between H^i and H^{i-1}. Then, we'll build a distinguisher C' for the G in the following way:

Input: $y \in \{0,1\}^{2n}$ (y either comes from U_{2n} or from $\mathsf{G}(U_n)$)

Operation: Run the oracle as usual. However when getting to the i^{th} "fake invocation". In this invocation it is supposed to take an internal node v which is marked "to be chosen at random", and choose a random value x for it. In the hybrid H^{i-1} the oracle chooses $(x_0, x_1) = \mathsf{G}(x)$ and uses that to label v's children, then erasing x. In the hybrid H^i the oracle chooses x_0 and x_1 at random. Our distinguisher will simply let $(x_0, x_1) = y$ and use this as the labeling.

It is clear that if $y \sim \mathsf{G}(U_n)$ then we get H^{i-1} and if $y \sim U_{2n}$ we get H^i. Therefore the success of C' in distinguishing $\mathsf{G}(U_n)$ and U_{2n} equals the success of C in distinguishing H^{i-1} and H^i. Since C' is only polynomially slower than C we're done. □

□