CS 4501-6501 Topics in Cryptography 16 Feb 2018

Lecture 5, CPA Secure Encryption from PRFs

Lecturer: Mohammad Mahmoody *Scribe: J. Fu, D. Anderson, W. Chao, and Y. Yu*

1 Review

Ralling: CPA Security and Randomized Encryption. Last time, we discussed CPA Security and how it allows more power to the adversary compared to the "single-message" indistinguishability-based definitions. The security goal here is that, as long as the adversary is computationally bounded (to polynomial time computation), they cannot guess the message that is being encrypted. More formally, our goal is to show that for all polynomial-time adversaries,

$$|\Pr_{\text{World 0}}[b' = 1] - \Pr_{\text{World 1}}[b' = 1]| \leq \text{negl}(n).$$

where in World b, the challenger always encrypts m_b (and m_0, m_1 are the two messages proposed by the adversary to be encrypted), and it is crucial that the adversary has access to an encryption oracle all along the security game. Namely, he can request any message m to be encrypted for him.

In other word, in different worlds where different message is encrypted and returned to the adversary, the probability of adversary's output bit is extremely close (i.e., at most far by a negligible amount). Also note that, in this way of writing the definition, we no longer call the output b' a "correct" or "wrong" guess. In PS1 you proved that the other way of defining the security in which the message m_b is chosen at random, and adversary tires to guess b is equivalent to the above definition.

However, if we want to prove an encryption algorithm to be secure under the CPA security definition, we need to use randomness in the encryption. The reason is that, otherwise, the adversary can request m_0, m_1 both to be encrypted for him by the encryption oracle and he can then try to see which one of them matches the challenge given to him in the security game. So the question is: how do we use randomness *pro pertly* to achieve CPA security?

Recalling: Usefulness of PRGs for Single-Message Security. Recall how we used PRGs to encrypt messages that are longer than keys and achieve single-message indistinguishability security against computationally bounded adversaries. The idea was to start from a key k whose length is the security parameter n, and then use the PRG g to expand k into n^2 bits as follows $g(k) = K$, and then we can use the longer K and do a "one-time-pad".

Now, one might ask: how can we use similar ideas to encrypt multiple messages $m_1, m_2, ...$ over a period of time securely. Namely, suppose m_1 is the message that we want to encrypt today, m_2 is the message we want to encrypt tomorrow, etc.

A cheating idea above is to split the long key K into multiple pieces $k_1, k_2, \ldots$ and use k_i to mask m_i. There are two problems with this approach: one is that Alice and Bob (the encrypter and the decrypter) need to have a "state" and be "synchronized" because they have to keep track of the length of the prefix of K that is used so far. Secondly, we might not know ahead of the time the total length of the messages $m_1, m_2\ldots$ that will be encrypted over time (if we knew, we could use the PRG to expand the initial k long enough). The second idea could be resolved by using the idea of "stream-ciphers" (see the Katz-Lindell book for an in depth discussion).

However the synchronization is not ideal, and so for now, we focus on only encrypting a single message.

The way we proved the security of single-message secure encryption was to show that $g(k) \oplus m$ is indistinguishable from U_ℓ where $\ell = |m| = |g(k)|$ is the length of the message and the output of PRG. But how exactly we prove this? We used a hybrid argument by using a hybrid world in which the adversary obtains $m \oplus U_\ell$ where U_ℓ is *truly* randomly. The idea was that: if an adversary can distinguish $m \oplus U_\ell$ from $m \oplus g(k)$, we can turn this into an attack against g as follows: given a string z we will give $z \oplus m$ to the attacker who distinguishes $m \oplus U_\ell$ from $m \oplus g(k)$ and we will also distinguish $g(U_n)$ from U_ℓ by the same advantage.

A useful abstract lemma. Here we describe an abstract lemma that was behind our argument. The good thing about this abstract lemma is that it can be used in broader settings to prove indistinguishability of two "efficiently processed" random variables as long as the original inputs are indistinguishable.

Lemma 1.1 (Indistinguishability of efficiently processed indistinguishable data)**.** *Consider two distributions X_n, Y_n (each parameterized by n) that are indistinguishable from each other: $X_n \underset{c}{\approx} Y_n$ in eyes of any* $\text{poly}(n)$*-time adversary. Now suppose Z is a polynomial time (even potentially randomized) algorithm, and suppose $X'_n \equiv Z(X_n)$ be a random variable denoting the output of Z when it is given an input sampled from the distribution X_n. Similarly, let $Y'_n \equiv Z(Y_n)$ be a random variable denoting the output of Z when it is given an input sampled from the distribution Y_n. Then, it holds that the distributions X'_n and Y'_n are also computationally indistinguishable. Namely, for any polynomial time adversary A, there is a negligible function $\varepsilon(n)$ such that*

$$|\Pr[A(X'_n) = 1] - \Pr[A(Y'_n) = 1]| \le \varepsilon(n).$$

Proof. For sake of contradiction, suppose there is a polynomial time A for which

$$|\Pr[A(X'_n) = 1] - \Pr[A(Y'_n) = 1]| > \delta(n)$$

for some non-negligible function $\delta(n)$. Then we can use A and Z to distinguish X_n from Y_n as follows using algorithm B. Given any input w (which we don't know if it is generated by X_n or Y_n), B does the following: output whatever $A(Z(w))$ outputs.

Note that if $w \leftarrow X_n$ is generated from X_n, then the input $u = Z(w)$ that we give to A is distributed according to X'_n, and if $w \leftarrow Y_n$ is generated from Y_n, then the input $u = Z(w)$ that we give to A is distributed according to Y'_n, therefore the *polynomial* time algorithm $B(\cdot) = A(Z(\cdot))$ will achieve:

$$|\Pr[B(X_n) = 1] - \Pr[B(Y_n) = 1]| > \delta(n)$$

as well for the same non-negligible $\delta(n)$ which is a contradiction, because it shows that X_n and Y_n are indeed distinguishable by a polynomial time adversary.

Note that in the above proof it was crucial that Z was polynomial time. $\square$

2 Pseudorandom Functions

Pseudorandom function is like a very strong pseudorandom generator with a super long output (formalized as below) so long that it is not even possible to read it in polynomial time! But we can read chunks of it by giving an "input" to the "function" that computes the "specified block" in the output.

Informally, consider the following levels of "good" PRGs (the last one is informal).

PRGs: From Worst to Best

- Good: A PRG that has input k_n with length n and outputs a K_{2n} with length of $2n$ that is indistinguishable from truly random U_{2n}. In short, $K_{2n} \underset{c}{\approx} U_{2n}$.

- Better: A PRG that has input length n and outputs a larger length, e.g. n^{100}. Then there is $K_{n^{100}} \underset{c}{\approx} U_{n^{100}}$ in poly(n).

- Best: A PRG that has input length n and outputs a length of $\approx n \times 2^n$, namely 2^n "blocks" of length n.

Problem. The third item above is unrealistic. The problem here is that the algorithm does not have time to read the entire input in polynomial time. Thus, we must find a way to compute each segment separately. So we can think of the "original" input k as a privately kept key, and the "new" input i that indicates the block that we want to read from the output as the "new/actual" input.

The idea is to have a keyed function that is indistinguishable from a truly random function (just like the output of PRG is indistinguishable from truly random output). We will then use PRFs to achieve CPA secure encryption. More formally, consider the set of all functions $f : \{0,1\}^n \to \{0,1\}^n$, then we know that there are $(2^n)^{2^n}$ such functions. In order to describe a random function in this set, we would need $n \times 2^n$ bits. Informally, a pseudorandom function cannot be distinguished by a random function that has even exponentially man random bits.

The actual definition of PRF is quite simple. Below, we define a version in which the output is of length ℓ.

Definition 2.1 (Pseudorandom Function (PRF))**.** A PRF $F\colon \{0,1\}^n \times \{0,1\}^* \mapsto \{0,1\}^\ell$ is a function that takes in a key of length n and an arbitrary length input x, and outputs an output of length ℓ. The output length ℓ could be a constant, or parameterized by n (e.g., to be equal to n).

The security definition of F requires that no polynomial time algorithm can distinguish between the following two worlds:

- *oracle* access to $F(k,\cdot)$ where k is chosen at random.
- *oracle* access to a randomly chosen function f that maps strings to ℓ bits.

More formally, for all polynomial time algorithm D there is is a negligible function $\text{negl}(n)$ such that:

$$|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]| \leq \text{negl}(n).$$

Some texts define other variants of PRFs by default, e.g., by asking the output to be of the same length of the key k, or even restricting the input length x to be also $n = |k|$ as the length of the output.

Do it yourself: Try to show that all variants discussed above are equivalent (one can be used to derive the other one). Problem 3.10 from the book which is part of the PS2 can be an inspiring technique.

Note that the security of a PRG is different than the security of a PRF, but they are similar in many ways. You can compare the secretly kept input of the PRG to the secret key of a PRF and the long output of PRF (on all inputs) could be compared to (relatively long) output of PRG.

How do we Obtain Pseudorandom Functions?

- **(1) Using Length-Doubling PRGs.**

$$PRG : n - bit \longrightarrow 2n - bit \text{ can be used to } \Longrightarrow PRF$$

See the book for more details. We sketch the ideas here for completeness. It shows that PRF, as a computational assumption, is not stronger than PRGs, as PRGs can be used to obtain PRFs. Given key k and input $x \in \{0,1\}^n$, use a tree-based idea with PRGs at every node in the tree until a leaf node is reached. With the tree-based idea, we start at the root node, which contains a PRG that either tells us to go left or right down the tree until we get to a leaf node, which determines the segment. We will now formally construct a pseudorandom function from a pseudorandom generator. The following construction will produce a binary tree, where n is the depth of the tree.

Assume there exists a pseudorandom generator G that doubles the size of the input (i.e. $G : \{0,1\}^n \to \{0,1\}^{2n}$). We let $G_0(x)$ be the first n bits of $G(x)$, and $G_1(x)$ be the last n bits of $G(x)$. The length-preserving pseudorandom function F will be constructed such that $F_i : \{0,1\}^n \to \{0,1\}^n$:

$$F_k(x) = F_k(x_1 x_2 ... x_n) := G_{x_n}(G_{x_{n-1}}(...(G_{x_1}(k))...)).$$

This construction can be *proved* to be secure and the proof makes a very interesting use of the hybrid-argument technique. You can find the proof in the book.

However, the more efficient and practical way of obtaining PRFs is to use secure hash functions in a heuristic way. So, here we do not prove the security of PRFs based on a specific property of hash functions, but the "good" hash hash functions in practice lead to PRFs that we don't know how to break.

- **(2) Using Hash Functions.** A simple way of getting PRFs that are pretty good in practice is to let

$$F(k, x) = H(k||x)$$

where H is a "secure" hash function like SHA3 or SHA-256 and $||$ is the string concatenation. In the latter case, the output will be 256 bits, but SHA3 allows the output length to be chosen. So, even though the code of H is public and everyone can run it, beause we are plugging in a fixed random key as part of the prefix to the input, not having k we cannot guess the output $F(k, x)$ and if we have oracle access to it, no one knows how to distinguish it from truly random oracle in a polynomial time.

In other words, without the key, SHA-256 is NOT a PRF, it is just a deterministic hash function, as everyone knows how to compute it. But we can still use hash functions that do not have a key by concatenating the key to Y and hashing the concatenation, which will still allow it to appear random.

A good hash function should pass many tests (like being "collision-resistant") but we will talk about these properties later one when we need them.

3 CPA secure encryption using PRFs

To achieve CPA secure encryption, without loss of generality, it is enough to only encrypt one block (e.g. $M = \{0,1\}^l$). This is something you prove in the problem set 2 and formalized in the following claim.

Claim 3.1. Enc(k, x) *where* $x = (x_1, x_2, ..., x_k)$ *and each* x_i *is* ℓ *bits. Let* y_i *be a fresh encryption* ($y_i \longleftarrow \text{Enc}(k, x_k)$) *using fresh randomness, and then output* $(y_1, ..., y_k)$ *as the ciphertext. To decrypt, decrypt each block* y_i *separately and put the pieces together to get back* x *If* Enc *is CPA-secure for a fixed block length* ℓ*, then it is CPA secure for all messages* $\{0,1\}^*$ *if we encrypt and decrypt them as described above.*

Examples. Suppose we have a pseudorandom function such that $F(k,x) = y$, where k is n-bits, x has arbitrary length, y is l-bits, and F is a PRF. Goal: obtaining $\text{Enc}(k,m;r) \longrightarrow c$ and $\text{Dec}(k,c) \longrightarrow m$, where k is n-bits and $m;r$ is l-bits in a CPA secure way. We go over some efforts, each time we fail, but we eventually put all the ideas together and it works!

1. $\text{Enc}(k,m) = F(k,m)$. This is not a good way to encrypt, because the decryption method is not clear at all! We don't know how to go back from $F(k,m)$ to m even if we know k.
2. $\text{Enc}(k,m) = F(k,m) \oplus m$. This also has the same issue, but it just has a more complicated encryption!
3. $\text{Enc}(k,m) = F(k,1234) \oplus m$ and $\text{Dec}(k,c) = F(k,1234) \oplus c$. Note that 1234 is a constraint input. This decryption method is correct and even the decryption is also efficient and well defined! The only problem is that we do not have CPA security. (Note, we have not injected more randomness into the encryption, and we already know that deterministic encryption cannot be CPA secure as we saw last time..)
4. $\text{Enc}(k,m,r) = F(k,r) \oplus m$ and $\text{Dec}(k,c) = F(k,r) \oplus c = m$. This does not work either because r is not in the decryption algorithm. If it was given to the decryptor as well, it would be just part of the key! All we have to do is to twist this scheme a bit more!

Constructing a CPA-secure encryption from any PRF:
Let $F(\cdot,\cdot)$ be a secure pseudorandom function with output length ℓ, then define a private-key encryption scheme for messages of length ell as follows:

1. **Gen:** on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it
2. **Enc:** on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^\ell$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext:
$$c = [r, m \oplus F_k(r)]$$
3. **Dec:** on input a key $k \in \{0,1\}^n$ and a ciphertext $c = [r,y]$, output the plaintext message
$$m = y \oplus F_k(r)$$

.

Proof of Security. The basic idea is that we analyze the security of the scheme in an idealized world where a truly random function $R(\cdot)$ is used in place of PRF $F_k(\cdot)$, and show that the scheme is secure in this case. Next, we claim that if the scheme were insecure when $F_k(\cdot)$ was used this would imply the possibility of distinguishing $F_k(\cdot)$ from a truly random function.

We begin by constructing real and ideal worlds as follows:

1. **Real World 0:** Challenger is given encryption oracle $\mathrm{Enc}(k, \cdot)$ and receive messages m_0 and m_1 all of the same lengths. Challenger sends back an fresh encryption of m_0: $\mathrm{Enc}(k, m_0)$, and adversary A outputs a bit b and
$$p_0 = \Pr[b = 1].$$

2. **Ideal World 0:** Let its encryption scheme be exactly the same except that a truly random function $R(\cdot)$ used in place of $F_k(\cdot)$, so adversary A receives an fresh encryption of m_0: $\widetilde{\mathrm{Enc}}(m_0) = (r, R(r) \oplus m)$, and whenever there is an encryption request for a message m, $\widetilde{\mathrm{Enc}}(m) = (r, , R(r) \oplus m)$ is returned (with fresh randomness r). Adversary A outputs a bit b and
$$q_0 = \Pr[b = 1].$$

3. **Ideal World 1:** This world is exactly the same as Ideal world 0, the only difference is that m_1 is "encrypted" for the adversary and
$$q_1 = \Pr[b = 1].$$

4. **Real World 1:** This world is exactly the same as Real world 0, but adversary receives $\mathrm{Enc}(k, m_1)$ with output b:
$$p_1 = \Pr[b = 1].$$

The indistinguishability of Real World 0 and Real World 1 follows from the following two claims.

Claim 3.2. *If F is a secure PRF, then $|p_0 - q_0|$ and $|p_1 - q_1|$ are both negligible for any polynomial time adversary.*

Proof. We prove this for $|p_0 - q_0| \leq \mathrm{negl}(n)$ and the other part is similar. The reason is that if $|p_0 - q_0| \geq \varepsilon(n)$ for some adversary A, then we can use A to distinguish between F and real random function $R(\cdot)$. That is because, we can run A and whenever it wants to call its oracle $O \in \{R, F\}$, we would use our oracle O' (which we want to find out if it is random or pseudorandom) and provide the "encryption" $(r, O'(r) \oplus m$ for A. This holds both for the encryption oracle and the encrypted challenge m_0. Now, if our oracle O' is F, we end up simulating the world Real 0 for A, and if O' happens to be truly random, we simulate the world Real 0 for A. □

Claim 3.3. *If $2^{-|r|}$ is negligible (e.g., we can choose the length of r to be n or even as small as $\omega(\log n)$) then $|q_0 - q_1|$ is $\mathrm{negl}(n)$.*

Proof. When we run either of the experiments Ideal 0 or Ideal 1, as long as none of the randomness choices r collide for any of the encrypted messages and that of the encrypted challenge m_b (which is encrypting different messages in these two experiments) the message that the adversary receives as challenge looks completely random and *independent* of everything else that he has received. Note that if the adversary gets lucky and the randomness r

that is used for encrypting m_b gets equal to another encryption that he gets from its oracle, then he can clearly tell apart which world he is in, but the point is that this is the *only* thing that might be different between the two worlds. Therefore, if we choose r large enough, the probability of k encryption queries having a similar randomness with that of the encryption of m_b would be at most $k/2^{|r|}$ where $k = \text{poly}(n)$. Therefore, if $1/2^{|r|}$ happens to be negligible, this probability is very small and then with probability $1 - \text{negl}(n)$ the two experiments Ideal 0 and Ideal 1 would be the "same" in eyse of the adversary in an statistical sense (so the output bits would be negligibly close). $\square$