CS 4501-6501 Topics in Cryptography 2 Feb 2018

Lecture 3: Computational Secrecy and Pseudorandom Generators

Lecturer: Mohammad Mahmoody, Scribe: Lahiru Wijayasingh, Austin Chen, Curtis Kahn

1 Introduction

Last lecture, we dealt with perfectly secret/secure encryption schemes. A perfectly secure (i.e., information-theoretically secure) encryption scheme cannot be broken by adversaries who have unlimited time and computing power. But, as we saw, there is a limitation for perfectly secure encryption schemes: they require keys which are at least as long as the message that we are going to encrypt. To get around this problem we define *computational* secrecy. In real world scenarios, an adversary cannot have arbitrarily large computing power and infinite time. Any adversary has limited computation power and a limited amount of time to solve a problem. Computationally secure encryption schemes define methods to encrypt data so an adversary cannot break it within practical time limits, even with the fastest computers on Earth. These schemes may be called "practically unbreakable". Compared to information-theoretically secure schemes, computationally-secure schemes have the advantage of relying on much shorter keys. In other words, computationally secure schemes are weaker than information theoretically secure schemes, but they are secure enough for practical purposes.

Definition 1.1 ((t, ε) security)**.** An scheme is (t, ε)-secure if every adversary running for time at most t succeeds in breaking the scheme with "probability" at most ε.

What is meant by "breaking the scheme" differs with the exact security definitions involved. Ideally, the value of t should be greater than the time needed to break the scheme with feasible computing resources, and the probability value ε should be less than some predefined value of negligible probability value. We use the key length n as the "security parameter" and try to prove that for any feasible time $t(n)$ the adversary's chance is at most $\varepsilon(n)$ for a negligible $\varepsilon(n)$. But how should we define feasible and negligible?

Feasible time: We call $t(n)$ feasible, if $t(n) \leq n^c$ for some constant c (and sufficiently large $n > n_0$ for some constant n_0). Equivalently, we call $t(n)$ a polynomial function if it is feasible. We call $t(n)$ *super polynomial* if $t(n) \geq n^c$ for any constant c and sufficiently large n_0. So we would like to be secure against any adversary who runs in some super-polynomial time, and that will guarantee that all feasible running times are covered.

Negligible probability: We call $\varepsilon(n)$ negligible, if for all constant c there is some n_0 such that for all $n > n_0$, $\varepsilon(n) < 1/n^c$.

Check it yourself: if $t(n)$ is feasible, then $1/t(n)$ cannot be negligible. But $t(n)$ is super polynomial time if and only if $\varepsilon(n)$ is negligible.

Why was the boundary of feasible and negligible set to polynomials? It seems more like an arbitrary decision, but polynomials have nice properties (for example the are easy to work with because their composition is still a polynomial). In fact, and at the end of the day most private-key encryption algorithms are believed to be also exponentially secure.

Typical values can be $t = 2^{100}$ and $\varepsilon = 2^{-100}$.

In order to rely on computational hardness we need mathematical puzzles that are hard to solve, and then we can try to turn them into secure encryption algorithms!

1.1 Computationally Hard Puzzles

What do we mean by a hard problem? One that there does not exist an algorithm to solve it that has a "quick" run-time, where by quick we mean feasible/polynomial time as defined above.

Are there any puzzles that are computationally "hard"? It certainly seems so; for example, the Traveling Salesman Problem (TSP) is not known to be feasibly solvable (i.e., no polynomial (i.e., feasible) time algorithm is known for it, despite decades of research). The formal conjecture is that:

Conjecture: $\nexists$ poly-time algorithm for TSP that solves all instances correctly.

TSP is a member of the complexity class NP-Complete (NPC). All members of NPC are related to each other. If a poly-time algorithm exists for one, it exists for all. It is an open problem in cryptography of making an encryption scheme out of an NPC problem. (This means, an encryption algorithm whose security *only* relies on hardness of NPC problem.) If we could, it would be "stronger" than current encryption schemes.

Factoring large numbers into primes is also believed to be a hard problem (note that here we mean mathematically hard in a provable way, which is an open problem, but certainly no publicly published algorithm is known for this puzzle).

However, there are reasons to believe that factoring is not as hard as NPC problems. Every year factoring gets a little faster. More formally, there is no known "equivalence" relation between the hardness of factoring and TSP, while TSP is known to be equivalent (in terms of being feasibly solvable) to other NPC problems.

2 (t, ε) Indistinguishability Security for Encryption

We can model (t, ε) security with a security game:

1. Adversary selects $m_0, m_1 \in \mathcal{M} = \{0,1\}^m$
2. Challenger selects:
   (a) $k \leftarrow U_n$ (i.e., a uniform string of length n)
   (b) $b \leftarrow \{0,1\}$

(c) $\text{Enc}(k, m_b) \to c$

3. Adversary guesses b' and wins if $b = b'$

Definition 2.1. The scheme B is (t, ε) ind-secure if for all adversaries who run in time $t(n)$, the probability of winning in the game above is bounded to be at most $\frac{1+\varepsilon(n)}{2}$. We simply call B an ind-secure scheme if it is for all polynomial time $t(\cdot)$, there is a negligible $\varepsilon(\cdot)$ such that the scheme B is (t, ε) secure. (For example, if it was (t, ε) secure for $t(n) = n^{\log n}$ and $\varepsilon(n) = 1/t(n)$, then it would be secure.

3 Pseudo-randomness

Intuitively, pseudo-randomness means random in the eyes of the computationally bounded. What is a random string? One can model this question in two ways. First suppose, we are dealing with *fixed* strings.

Which of these sequences of 100 coin flips is more "random looking"?

$$S_1 = 0, 1, 0, 1, \ldots, 0, 1$$
$$S_2 = 0, 1, 1, 0, 0, 1, 0, 1, \ldots$$

Intuitively, S_2 looks more random, but also note that $\Pr[S_1] = \Pr[S_2] = 2^{-100}$ so how can we say S_2 is more random? The way Kolmogorov explained this was based on the complexity of generating each of these sequences. There is a short program outputting the first one, while, in order to generate the second one, we need to basically hard wire the sequence into the program's code..

Definition 3.1 (Kolmogorov Complexity)**.** The Kolmogorov complexity $K(x)$ of a string x is the minimum length of a program (in some fixed programming language, like Java) that would print x and stop.

It turns out the this definition is not very useful for cryptography, but is its own whole branch of study. In cryptography, we will work with *distributions* over strings. For example, a string that is generated as a ciphertext, and we want them to be as random looking as possible! In particular, we are interested in distributions that are *not* truely random, but they *look like* a random one.

Pseudorandom distributions. We will define pseudo randomness using a "security game". Given a string, guess whether it was generated truly randomly, or "differently". The goal should be to guess correctly with probability $> \frac{1}{2}$. If the adversary has no computational restrictions they can also guess correctly with probability $> \frac{1}{2}$. (The reason is that a non-uniform distribution would have statistical distance > 0 from the uniform one, and then there is a way to distinguish them, using the definition of statistical distance).

Now going back to our pseudorandomness game. Suppose $S_1 \equiv U_n$, namely, S_1 is a uniform truly random string of length n, and suppose we generate the second string S_2 with an efficient function $g(x) : \{0,1\}^{n/2} \to \{0,1\}^n$ by picking the the input of $g(\cdot)$ at random. Namely, $S_2 = g(U_{n/2})$. Informally, we call $g(U_{n/2})$ pseudoranom, if it is "hard" to distinguish it from truely random distribution U_n.

The following is a formal definition of a pseudo random generator (PRG):

Definition 3.2 (Pseudo random generator)**.** Suppose g is an efficient (ie. polynomital time) algorithm that takes x and outputs a string of length $2 \cdot |x|$. We call g a pseudorandom generator (PRG for short) if for all poly-time adversaries A, there is a negligible function ε such that $\Pr[\text{Adv wins}] \le \frac{1+\varepsilon(n)}{2}$ in the security game below.

Construction 3.3 (Security game for PRG's)**.** :

1. Challenger chooses: $b \leftarrow \{0,1\}$
   If $b = 0$: then $\quad y \leftarrow U_n$
   Else (i.e., if $b = 1$) $\quad y \leftarrow g(U_{n/2})$
   Challenger sends y to the adversary
2. The adversary outputs a guess bit b' and wins if $b = b'$

Note that g cannot be truly random since the total number of U_n outputs is $|U_n| = 2^n$ while the possible outputs of g is $|g(U_{n/2})| \le 2^{\frac{n}{2}} \ll 2^n$.

But how can we actually win in the security game above, if you are not computationally bounded? One possible strategy is this: generate the set $S = \{x | x \in g(U_{n/2})\}$. If $y \in S$ output $b' = 1$, otherwise output $b' = 0$. The size of S is $|S| \le 2^{\frac{n}{2}}$, which is still not poly-time, so this is going to take a lot of time!

Note that if you can easily invert g, the adversary can also use that strategy to win with probability better than $1/2$ even with computational bounds. So a secure PRG implicitly has to be "one way" as well.

Here our goal goal is to construct an indistinguishable secure private key encryption scheme out of a PRG where the encryption works on length n and the key is of length $n/2$ (recall that this was impossible using perfect security definition).

Construction 3.4. Encryption scheme B using PRG g: Given a key k of length $n/2$ we can encrypt messages of length n as follows
$\text{Enc}(k,m) = g(k) \oplus m$
$\text{Dec}(k,c) = g(k) \oplus c$

Completeness property is satisfied: $\text{Dec}(k, \text{Enc}(k,m)) = m \oplus g(k) \oplus g(k) = m$. The main question is whether is B secure or not.

4 Proof of Security

This is the first formal proof of a schemes security in class, so it is important to figure our all the details!

Theorem 4.1. *If g is a secure PRG, then B is a secure secret key encryption scheme.*

This proof will use a useful trick in transforming security games.
Original security game for PRG's:

1. Challenger chooses: $b \leftarrow \{0, 1\}$
   If $b = 0$: then $\quad y \leftarrow U_n$
   Else (i.e., if $b = 1$) $\quad y \leftarrow g(U_{n/2})$
   Challenger sends y to the adversary

2. The adversary outputs a guess bit b' and wins if $b = b'$

and we have $\Pr[\text{Adv wins}] \leq \frac{1+\varepsilon(n)}{2}$ for a negligible ε.

We can write the same security definition based on a game about worlds:

World 0	World 1
$c \leftarrow \text{Enc}(k, x_0)$ for random k	$c \leftarrow \text{Enc}(k, x_1)$ for random k
Challenger gives c to ADV	Challenger gives c to ADV
Adv outputs guess b'	Adv outputs guess b'

There is no probability of 'winning' anymore, but we call the scheme secure if

$$\left| \Pr_{\text{world } 0}[\text{ADV}[c] = 1] - \Pr_{\text{world } 1}[\text{ADV}[c] = 1] \right| \leq \varepsilon(n)$$

for a negligible ε. (In the first problem set, you can see why this is an equivalent definition).

The next step in the proof is a common proof technique that we will get very used to, and it is called the hybrid argument:

World 0	Hybrid 0	Hybrid 1	World 1
CHAL $\xleftarrow{m_0, m_1}$ ADV	CHAL $\xleftarrow{m_0, m_1}$ ADV	CHAL $\xleftarrow{m_0, m_1}$ ADV	CHAL $\xleftarrow{m_0, m_1}$ ADV
CHAL $\xrightarrow{m_0 \oplus g(U_{n/2})}$ ADV	CHAL $\xrightarrow{m_0 \oplus U_n}$ ADV	CHAL $\xrightarrow{m_1 \oplus U_n}$ ADV	CHAL $\xrightarrow{m_1 \oplus g(U_{n/2})}$ ADV
$\Pr[\text{ADV} \to 1] = p_0$	$\Pr[\text{ADV} \to 1] = p'_0$	$\Pr[\text{ADV} \to 1] = p'_1$	$\Pr[\text{ADV} \to 1] = p_1$

Note that: World 0 and World 1 have pseudo random key generators, while Hybrid 0 and Hybrid 1 has random key generators;

Goal. We will prove that if the PRG is computable in time s (which is a polynomial) and it is (t, ε) for some super-polynomial t and negligible ε, then, the encryption scheme is also (t', ε') secure for $t' \approx t - s$ and $\varepsilon' \approx 2\varepsilon$.

Do it yourself. Show that if we achieve the above goal, then the security of PRG implies the security of the encryption algorithm. (Hint, if t, ε are are super-polynomial/negligible, so would be t', ε'.)

We want to prove that $|p_0 - p_1| \leq \varepsilon$. We will show that $p'_0 = p'_1$ and that $|p_0 - p'_0| \leq \varepsilon$ and that $|p_1 - p'_1| \leq \varepsilon$, and then the goal follows by the triangle inequality.

Hybrid 0 and Hybrid 1 give the adversary a message that is encrypted using the one-time pad. So, in eyes of the adversary, he cannot distinguish them better than advantage 0 (because one time pad is perfectly secure) and so the probability of outputting one in these two "experiments" is the same.

Using the security of the PRG. Note that so far we have not used the security of the PRG. Intuitively, because we are using a PRG g in World 0, the adversary should not notice the change when we switch from g to fully random string in Hybrid 0 (because g is a PRG). But how much the adversary can actually notice this? This needs a formal argument, aka a seucrity proof or a security reduction. Suppose for sake of contradiction that there is an adversary A running in time $t' = t - s$ that can achieve $|p_0 - p_1| \geq \varepsilon$. Then, can we use this adversary to come up with another adversary A' that runs in time t and breaks the security of the PRG? The answer is yes! This is how A' will work:

Given a string y, which A' does not know if it is generated as yU_n or $y \leftarrow g(U_{n/2})$, the adversary can use y to encrypt message m_0 for itself!! namely it can get $c = m_0 \oplus y$, and then it can run $A(c)$ to see what A outputs. Now, if it was the case that $y \leftarrow U_n$, then A (and consequently A') will output 1 with probability p'_0 (because A' is running A in experiment Hybird 0) and if it was the case that $y \leftarrow g(U_{n/2})$, then A (and consequently A') will output 1 with probability p_0 (because A' is running A in experiment World 0). So, if A was successful in ε distinguishing a random from a pseudo-random string, A' will have the same advantage in distinguishing between World 0 and Hybrid 0.

What is the running time of A compared to A'? A' runs A once on an input c that it prepares as $c = \text{Enc}(.)$ that involves running g, so if the running time of computing g is some polynomial s, so the running time of A' would be $\approx t + s$.