Lecture 3
JANUARY 26

Instructor: Fang Song

Disclaimer. Draft note. No guarantee on completeness nor soundness. Read with caution, and shoot me an email at fsong@pdx.edu for corrections/comments (they are always welcome!)

Logistics. HW 2 Problem 1(c) typo. $k = -^{\ell}$ should be $k = 0^{\ell}$.
Last time. Perfect secrecy: equivalence, example (one-time pad), limits. Computational secrecy.
Today. Pseudorandom generators and stream ciphers. Block ciphers intro.

FS NOTE: Use *attacker, adversary* (and *eavesdropper* till we introduce CPA) interchangeably.

We will introduce two types of ciphers that achieve computational secrecy. Recall the definition of *computational secrecy*. Informally, it says that "any efficient (PPT) adversary can not tell apart the ciphtertexts of two messages except with negligible probability". We formalize it in an experiment/game.

1. Adversary is given input 1^n, and $\mathscr{A}$ outputs a pair of messages m_0, m_1 with $|m_0| = |m_1|$.
2. CH generates a key $k \leftarrow G(1^n)$, and a uniform $b \leftarrow \{0,1\}$. Compute *challenge ciphertext* $c \leftarrow E_k(m_b)$ and give to $\mathscr{A}$.
3. $\mathscr{A}$ outputs a bit b' as the guess of b.
4. Define $\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n)$ the output of the experiment to be 1 if $b' = b$, and 0 otherwise. We call $\mathscr{A}$ succeeds if $\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n) = 1$.

Figure 1: Adversarial indistinguishability experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n)$

Computational secrecy says that $\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n) = 1] \leq 1/2 + \mathrm{negl}(n)$ for any PPT Adv. Namely no efficient adversary can do better than an essentially random guess. Note: subscript specifies adversary and the scheme.

1 Stream Ciphers & Pseudorandom generators

How to construct a computationally secret cipher? Recall our hope was to encrypt a possibly long message with a "short" key. Where to start? Well, cryptographers are kinda lazy. They usually start by adapting or fixing existing constructions. So far we've seen OTP which is perfectly secret with a random key: $E_k(m) : c := k \oplus m$. We know unfortunately the key has to be as long as the message, but is it possible to start with a short random key (call it a *seed*), and expand it to a long "random" string? Of course we can not hope for the resulting string to be truly random (why not?), but maybe it suffices for the string to "*look*" random, as least as far as an efficient adversary is concerned. This motivated people to define the notion of a *pseudorandom generator* (PRG in short), and the encryption scheme by plugging a pseudorandom key to OTP is usually called a stream cipher in practice. For this course, we may abuse the terminology and identify PRGs and stream ciphers.

Let's define a PRG formally:

- $\ell(n)$: a polynomial $\ell(n) > n$. Call it the *expansion factor*.
- $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$: *deterministic* polynomial-time algorithm.

Definition 1. We say G is a pseurandom generator (PRG) if

1. (**Expansion**) $\ell(n) > n$ for every n.
2. (**Pseudorandomness**) for any PPT $\mathscr{A}$, $\Pr\left[\mathsf{PRG}_{\mathscr{A},G}(n) = 1\right] \le \frac{1}{2} + \text{negl}(n)$.

How to define "look" random? Again, we can think of a distinguishing game (Fig. 2):

1. CH generates a uniform bit $b \leftarrow \{0,1\}$.
   - if $b = 0$, choose $r \leftarrow \{0,1\}^{\ell(n)}$ uniformly at random.
   - if $b = 1$, choose a random seed $s \leftarrow \{0,1\}^n$ and compute $r := G(s)$.
2. Adversary is given input 1^n and r. $\mathscr{A}$ outputs a bit b' as the guess of b.
3. Define $\mathsf{PRG}_{\mathscr{A},G}(n)$ the output of the experiment to be 1 if $b' = b$, and 0 otherwise. We call $\mathscr{A}$ succeeds if $\mathsf{PRG}_{\mathscr{A},G}(n) = 1$.

Figure 2: PRG indistinguishability experiment $\mathsf{PRG}_{\mathscr{A},G}(n)$

We usually call such an adversary a distinguisher and use distinguisher and adversary/attacker interchangeably in this context. We often call the output from a pseudorandom generator a pseudorandom string, although this is totally nonsense[1].

From a slightly different perspective, we can imagine that an adversary as performing some efficient statistical test and output one bit indicating if an input string has passed the test (e.g., does the string has more than 10 consecutive 0s). We want that for any efficient test, a pseudorandom string is equally likely to pass it as a truly random string would (except for small discrepancy). Therefore can give an alternate formulation of the pseudorandom condition: for any efficient distinguisher D,

$$\left|\Pr[D(G(s)) = 1 : s \leftarrow \{0,1\}^n] - \Pr[D(r) = 1 : r \leftarrow \{0,1\}^{\ell(n)}]\right| \le \text{negl}(n). \tag{1.1}$$

Eqn 1.1 is what [KL: Def. 3.14] used. [KL: Exercise 3.5] Prove the two formulations are equivalent.

[1] Similarly, it makes no sense to say any fixed string is "random" though we often do so. Rather, pseudorandomness is a property of a *distribution* on strings.

1.1 Discussion on pesudorandomness

PRG is impossible against *unbounded* attackers.) Suppose $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ is a PRG, consider the following distinguisher

Given $r \in \{0,1\}^{\ell(n)}$, D does

1. Exhaustively search $s \in \{0,1\}^n$ such that $G(s) = r$.
2. If there exists such an s, output 1. Otherwise, output 0.

Figure 3: An *inefficient* distinguisher for any PRG

The key observation is that the number of possible outputs of G is at most 2^n, yet there are $2^{\ell(n)}$ strings in total of length $\ell(n)$. Namely, G will only produce a tiny fraction of all ℓ-bit strings. Therefore, if $r = G(s)$ (i.e., generated by PRG), $\Pr[D(G(s)) = 1] = 1$. But if $r \leftarrow \{0,1\}^{\ell(n)}$ is a uniformly random string, $\Pr[D(r) = 1] = \Pr[r \in \{G(s) : s \in \{0,1\}^n\}] = \frac{2^n}{2^{\ell(n)}} = \frac{1}{2^{\ell(n)-n}}$. Hence

$$\Pr[D(G(s)) = 1] = 1 - \Pr[D(r) = 1] \geq 1 - \frac{1}{2^{\ell(n)-n}},$$

which is 1/2 even for the least ambitious goal of getting one-bit surplus ($\ell(n) = n+1$). As $\ell(n)$ grows, the difference (break of PRG) gets bigger and bigger.

Lesson: has to restrict to efficient distinguishers only.

Seed length. Seed is analogous to the secret key in an encryption scheme. It must be sufficiently long to avoid *brute-force* attack.

Existence of PRGs. Unfortunately, we do not know yet how to prove existence of PRGs *unconditionally*, although we have strong evidence to believe so. We will talk more later. For now, let's state the first important assumption in this course.

Assumption 2. *There exists a PRG $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ with expansion factor $\ell(n) > n$.*

1.2 PRG to comp. secret encryption

Now that (we assume that) we have a PRG at hand, let's complete our goal in the beginning and give a computationally secret encryption scheme with shorter keys.

Correctness easy to verify.

Theorem 3 (KL-Thm.3.18)**.** Π *is computationally secret (i.e. has indistinguishable encryptions in the presence of an eavesdropper), assuming Assumption 2.*

Proof by REDUCTION. Here we introuduce a general framework for security proofs in modern cryptography. (An extremely simple but useful technique called *hybrid argument* is also implicit in the proof to come, but we will treat it more carefully in a future lecture). We do not prove unconditionally that a construction is secure, but rather, we assume some *lowe-level* primitive

Given PRG $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$. Security parameter 1^n. Construct $\Pi = (KG, E, D)$:

- KG: choose uniform key $k \leftarrow \{0,1\}^n$.
- E: $c = E_k(m) := G(k) \oplus m$.
- D: $m = D_k(m) := G(k) \oplus c$.

Figure 4: A fixted-length comp. secret encryption from PRG

or problem is secure or hard to solve, and then prove the construction is secure *under* this assumption. We do so by presenting an explicit REDUCTION, which transforms any efficient adversary $\mathscr{A}$ that succeeds in "breaking" the construction into annother efficient algorithm $\mathscr{A}'$ that sovles the problem that was assumed to be hard. This will demontrate a contradiction. It's often helpful to draw a redcution diagram:

FS NOTE: Draw reduction diagram

To recap: to prove

$$\boxed{\text{if assumption } S \text{ holds, then scheme } \Pi \text{ is secure;}}$$

we use a reduction to show the *contrapositive*

$$\boxed{\text{if one breaks } \Pi\text{, then one can also violate the assumption } S.}$$

Proof. Proof of Thm. 3 Assume there is an PPT adversary $\mathscr{A}$ such that $\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n) = 1/2 + \varepsilon(n)$, for $\varepsilon(n) \geq 1/p(n)$ for some polynomial p. We construct a distinguisher

FS NOTE: Draw picture

Distinguisher D: Given input string $w \in \{0,1\}^{\ell(n)}$

1. Run $\mathscr{A}(1^n)$ to obtain a pair of messages $m_0, m_1 \in \{0,1\}^{\ell(n)}$
2. Choose $b \leftarrow \{0,1\}$. Set $c := w \oplus m_b$.
3. Give c to $\mathscr{A}$ and obtain b'. Output 1 if $b' = b$, and 0 otherwise.

We will demontrate that

$$\left| \Pr[D(w) = 1 : w \leftarrow \{0,1\}^{\ell(n)}] - \Pr[D(w) = 1 : w = G(s), s \leftarrow \{0,1\}^n] \right| \geq \varepsilon(n),$$

which shows a contradiction since it is supposed to be negligible by Assumption 2.

- $w \leftarrow \{0,1\}^{\ell(n)}$. If we look at what $\mathscr{A}$ sees in the reduction, the challenge cipher $c := w \oplus m_b$ is exactly what one gets from OTP with uniformly (long) random key w. We know that $\Pr[b' = b] = 1/2$ by the perfect secrecy of OTP, and therefore $\Pr[D(w) = 1 : w \leftarrow \{0,1\}^{\ell(n)}] = 1/2$.

- $w = G(s), s \leftarrow \{0,1\}^n$. Then $\mathscr{A}$ is playing exaclty the indistinguishablity game for Π, and hence $\Pr[D(w) = 1 : w = G(s), s \leftarrow \{0,1\}^n] = \Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathscr{A},\Pi}(n) = 1] = 1/2 + \varepsilon(n)$.

Therefore,

$$\left|\Pr[D(w) = 1 : w \leftarrow \{0,1\}^{\ell(n)}] - \Pr[D(w) = 1 : w = G(s), s \leftarrow \{0,1\}^n]\right| = \varepsilon(n) \geq 1/p(n)\,.$$

□

1.3 Constructing PRGs

General approaches for constructing PRGs. Why we believe in Assumption 2?

- We can construct secure PRGs under rather weak assumptions that *one-way* functions exist, or some computational problems can not be solved efficiently. More in a future lecture.
- We have candidate constructions in practice – stream ciphers (here).
- We can also get PRG from block ciphers (coming soon).

Getting more and more. *parallel* composition and *sequential* (Blum-Micali) composition.
Constructions in practice.

- Trivially insecure examples: $G : s \mapsto s\|s$, $G : s \mapsto s_0 s_{n-1} s_1 s_{n-2}, \dots s_{n-1} s_0$, etc.
- Examples that fail (easily): *Parity check PRG.*

$$G : s = s_0, \dots, s_{n-1} \mapsto s \| \oplus_{i=0}^{n-1} s_i\,.$$

  How do you distinguish it from a uniformly random string of length $n+1$?

- DVD encryption system: *linear feedback shift register (LFSR)* (also **easily breakable**). Parity check is a special linear function, and we can generalize to use general linear functions $f : s_0 \dots s_{n-1} \to \sum_{i=0}^{n-1} c_i \cdot s_i$ for $c_i \in \{0,1\}$ (e.g., Parity check f is the one with $c_i = 1$ for all i). In LFSR, s is treated as a state that keeps updating. $G(s)$ is computed by repeating the following $\ell(n)$ times. **FS NOTE**: draw diagram

$$s_0, \dots s_{n-1} \to s_1, \dots, s_{n_1}, f(s_0, \dots s_{n-1}),\ \text{output } s_0$$

Observing n cosecutive output bits, we can predict all future outputs if we know f (i.e. c_i). Even if we don't, we can find them by observing $2n$ bits output using linear algebra (Gaussian elimination).

- Good examples[2]: Trivium, RC4 (popular but vulnerabilities exist), subset-sum generator, etc.

2 Pseudorandom functions & Block ciphers

Now we come to the next type of ciphers: *block ciphers*. Block ciphers are the work horse in cryptography (espeically in private-key cryptography). Encryption, Message authentication, Hash functions... we will see many examples soon.

The abstract object is the so called *pseudorandom functions* (PRFs). Instead of considering "random-looking" strings, we cosider "random-looking" functions. Roughly, it means that it behaves the same as a truly random function, at least as far as an efficient observer is concerned. Again, it makes no sense to say any *fixed* function is pseudorandom (or random in any sense). It refers to a *distribution* on functions. But before we talk about pseudorandom functions, let's be clear what we mean by a *truly random function*.

Random functions. Consider $\mathscr{F} := \left\{f : \{0,1\}^n \to \{0,1\}^n\right\}$ be the collection of all possible functions that maps n-bit strings to n-bit strings. Similar to what we mean by a random string, a random function is just a sample from $\mathscr{F}$ uniformly at random. How many are there? Well if we think about the truth table of a function $f : \{0,1\}^n \to \{0,1\}^n$, for each input string x there are 2^n possible output strings we can map it to. Therefore

$$|\mathscr{F}| = \underbrace{2^n \cdot 2^n \cdot \ldots \cdot 2^n}_{2^n \text{ times}} = 2^{n2^n}.$$

This means you need $\Omega(\log|\mathscr{F}|) = \Omega(n2^n)$ bits to sample or write down the description of a random function.

There is a more intuitive and operational perspective of thinking about a random function. Imagine yourself as a machine implementing a random function, when an input x comes, you can just sample a $y \leftarrow \{0,1\}^n$ uniformly at random. The only thing you need to make sure is keep consistency i.e. give the same answer if an input comes again. This can be done by maintaining a lookup table for instance. This sample "on-the-fly" viewpoint will be useful for a lot of the analysis in the future.

[2]Read more on [Boneh-Shoup 3.7 - 3.9], and Handbook of Applied Cryptography (http://cacr.uwaterloo.ca/hac/) Chapter 6.