Winter 2018 CS 485/585 Introduction to Cryptography

Lecture 3

Portland State University
Lecturer: Fang Song

Jan. 16, 2018

Draft note. Version: January 25, 2018. Email fang.song@pdx.edu for comments and corrections.

Agenda

- (Last time) Perfect secrecy, One-time Pad, Limitations
- Limitations of Perfect secrecy: formal discussion
- Computational Secrecy: definition (via Indistinguishability Game)
- Stream Cipher, PRG, Proof by Reduction

Logistics

- D2L: forward d2l email to your email account. Solutions will also be posted on d2l

Limitations of perfect secrecy

Recall two observations that limit the applicability of OTP. Both are inherent limitations of *perfect secrecy*.

Limitation 1: Perfect secrecy cannot avoid long keys.

Theorem 1 (KL Theorem 2.10)**.** *Suppose $\Pi = (G, E, D)$ is perfectly secret, then $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof. Proof by contradiction. Suppose $|\mathcal{K}| < |\mathcal{M}|$. Consider a $c \in \mathcal{C}$, and we run the decryption algorithm on c under each $k \in \mathcal{K}$. Let

$$\mathcal{M}(c) := \{m : m = D_k(c) \text{ for some } k \in \mathcal{K}\} .$$

Clearly $\mathcal{M}(c) \leq |\mathcal{K}| < |\mathcal{M}|$. [1] This is already saying that given c, adversary can rule out some messages, which must not be possible by perfect secrecy. More precisely, consider the uniform distribution on $\mathcal{M}$, and some $m^* \notin \mathcal{M}(c)$. Then

$$\Pr[M = m^* | C = c] = 0 \neq \Pr[M = m] = \frac{1}{|\mathcal{M}|} .$$

□

[1] At most every key recovers a distinct message.

Limitation 2: "one-time" only. It is inherent too, but more general (applies to computational secrecy too). It has to do with the threat/attack model, i.e., secrecy against eavesdropping is not strong enough to ensure securely encrypting *multiple* messages under the same key.

Computational secrecy

Resolving limitation 1 To get away with *long* keys, have to appeal to relaxations in perfect secrecy.

- consider "efficient" attackers only
- accept "small" break of scheme

Both are necessary. Read [KL: page 51].

- Known-plaintext attack: exp. time, succ prob. almost 1.
- Key-guessing attack: constant-time, $1/|\mathcal{K}|$ success probability.

Review: perfect indistiguishability. Perfect secrecy can be read as the distributions (over $\mathcal{C}$) resulting from encrypting one message m and encrypting another message m' are *identical*. From the attackers point of view, these two distributions are *indistinguishable*. We can reformulate this by a *experiment* or *game* between an attacker/adversary $\mathcal{A}$ and the so-called *challenger* CH. This gives another equivalent definition of PS. [2]

[2] This will serve as a template for many of our definitions in the future.

Figure 1: Adversarial indistinguishability game $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$.

FS NOTE: Draw a game diagram

1. Adversary $\mathcal{A}$ outputs a pair of messages $m_0, m_1 \in \mathcal{M}$.
2. CH generates a key $k \leftarrow G$, and a uniform $b \leftarrow \{0,1\}$. Compute *challenge ciphertext* $c \leftarrow E_k(m_b)$ and give to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b' as the guess of b.
4. Define the output of the game to be 1 if $b' = b$, and 0 otherwise. Write $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi} = 1$ if the game output is 1, in which case we call $\mathcal{A}$ succeeds.

Definition 2 ([KL: Definition 2.5])**.** $\Pi = (G, E, D)$ is perfectly indistinguishable if for every attacker $\mathcal{A}$, it holds that

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}] = \frac{1}{2}. \tag{1}$$

Computational secrecy: The asymptotic approach Security parameter n – think of it as the key length, and everything is viewed as a function of n rather than concrete numbers: running time of all algorithms (including the adversary) and success probability.

"Efficient" = Probabilistic Polynomial Time (PPT), and "Small"= *negligible*.

Template for **asympototic** security
A scheme is *secure* if for every *probabilisitic polynomial-time* adversary $\mathcal{A}$ carrying out an formally specified attack, the probability that $\mathcal{A}$ *succeeds* in the attack is *negligible*.

Example 3. $2^{-n}, n^{10^{100}}, 2^{-\sqrt{n}}, n^{-\log n}, 100^{100 \log n}$ which ones are polynomially bounded and which ones are negligible?

Informally, no additional info. can be learned by *efficient* adversary except with *negligible* probability.

FS NOTE: Draw a game diagram

1. Adversary is given input 1^n, and $\mathcal{A}$ outputs a pair of messages m_0, m_1 with $|\mathbf{m_0}| = |\mathbf{m_1}|$.
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 $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b' as the guess of b.
4. Define random variable $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ the output of the game to be 1 if $b' = b$, and 0 otherwise. We call $\mathcal{A}$ wins if $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1$.

Figure 2: Adversarial indistinguishability game $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Distinctions from *perfect* indist:

- security parameter n.
- challenge messages $|m_0| = |m_1|$, otherwise no constraint on length (of course polynomial if $\mathcal{A}$ runs in polynomial time). [3]

[3] Encryption may not hide message length. Usually OK but not always (e.g., numerical data, 5 vs. 6-digit salary, database search). This is a price to pay for flexibility of encrypting arb-length messages. [KL: Ex 3.2 HW1]

Definition 4 ([KL: Def. 3.8])**.** A private-key encryption scheme $\Pi = (G, E, D)$ is *computationally secret* [4] if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function negl such that, for all n

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq 1/2 + \mathrm{negl}(n)\,,$$

where the probability is take nover the randomness of $\mathcal{A}$ and the randomness in the game (choosing the key and bit b, as well as any randomness in E). [5]

[4] or, as KL calls it, has *indistinguishable encryptions in the presence of an eavesdropper* (EAV-secure in short)

You might be wondering: can we adapt the other two equivalent formulations of perfect secrecy to computational secrecy? Sure!

[5] Be mindful of the two components of a security definition:

i. Security goal: indistinguishable of two ciphertexts except with negligible probability.
ii. Attack model: efficient attacker eavesdropping one ciphertext. We do not specify or restrict how the attacker instantiates its capability whatsoever.

- Analogue of [KL: Def.2.3] (no additional information) gives *semantic security*, the first definition for computational secrecy, proposed by Goldwasser and Micali. But it is much more complex and not intuitive to work with. (their paper proposed the indistinguishablity definition as well, but it was proven equivalent to semantic security in a later work [MicaliRacoffSloan].)
- Analogue of [KL: Eq 2.1 Lemma 2.4] (adversary behaves roughly the same on two ciphertexts): [KL: Def.3.9], equivalence to Def. 4 in [KL: Ex.3.4].

What does it mean that an adversary is *insecure* or broken? The negation of negligible advantage is a bit tricky. Recall negligible says ...

Stream Ciphers & Pseudorandom generators

How to construct a computationally secret cipher? *Stream cipher* and *Block cipher* are the two main types. We start with stream cipher in this lecture.

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 usually 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.

How to define "look" random? In reminiscent of [KL: Def.3.9] for computational secrecy, we would like an adversary behaves essentially the same on two possible inputs: output of a PRG or a string drawn uniformly at random. We can imagine that an adversary performs some efficient statistical test and outputs 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).

You may see "pseudorandom number generator" in the libraries of many programming languages. Most, if not all, do NOT meet the condition of a cryptogrpahic PRG. They are usually only designed to pass a few pre-determined statistical tests. In contrast, in the cryptogrpahic setting, the adversary could run any (efficient) test which is unknown to the designer ahead of time.

Definition 5. We say G is a pseurandom generator (PRG) if

1. (**Expansion**) $\ell(n) > n$ for every n.
2. (**Pseudorandomness**) for any efficient adversary D

$$\left| \Pr_{s \leftarrow \{0,1\}^n} [D(G(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{\ell(n)}} [D(r) = 1] \right| \le \mathrm{negl}(n)\,.$$

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[6].

[6] Similarly, it makes no sense to say any fixed string is "random" though we often do so. Rather, pseudorandom, as is uniformly random, refers to a *distribution* on strings.

Alternatively, we can think of a distinguishing game (Fig. 3):

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. $\mathcal{A}$ outputs a bit b' as the guess of b.
3. Define $\mathsf{PRG}_{\mathcal{A},G}(n)$ the output of the experiment to be 1 if $b' = b$, and 0 otherwise. We call $\mathcal{A}$ succeeds if $\mathsf{PRG}_{\mathcal{A},G}(n) = 1$.

Figure 3: PRG indistinguishability game $\mathsf{PRG}_{\mathcal{A},G}(n)$

Pseudorandomness is then defined by: for any PPT $\mathcal{A}$,

$$\Pr\left[\mathsf{PRG}_{\mathcal{A},G}(n) = 1\right] \le \frac{1}{2} + \mathrm{negl}(n)\,.$$

Prove the two formulations are equivalent. (HW 1 [KL: Exercise 3.5])

Existence of PRGs.

First of all, not surprisingly, 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 4: 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, $\Pr_{r\leftarrow\{0,1\}^{\ell(n)}}[D(r)=1] = \Pr[r \in \{G(s) : s \in \{0,1\}^n\}] = \frac{2^n}{2^{\ell(n)}} = \frac{1}{2^{\ell(n)-n}}$. Hence

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

It is $1/2$ even for the minimal need of getting one-bit surplus ($\ell(n) = n+1$). As $\ell(n)$ grows, the advantage (break of PRG) gets bigger and bigger.

Lesson: has to restrict to efficient distinguishers only.

Next, we look at a few simple ideas that fail.

- $G : s \mapsto s\|s$
- $G : s \mapsto s_0 s_{n-1} s_1 s_{n-2}, \dots s_{n-1} s_0$.
- *Parity check PRG*: $G : s = s_0, \dots, s_{n-1} \mapsto s\| \oplus_{i=0}^{n-1} s_i$.

It's not easy to get it right, but good PRGs do exist which we believe to be secure. We will come back to constructions of PRG, both in practice and theory. For now, let's introduce the first important assumption in this course.

Assumption 6. *There exists a PRG $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ with expansion factor $\ell(n) > n$.*

PRG to comp. secret encryption

Now that a PRG is at hand (by assumption), let's complete our goal in the beginning and construct a computationally secret encryption scheme with shorter keys.

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 5: A fixted-length comp. secret encryption from PRG(PRG-OTP).

Correctness is easy to verify.

Theorem 7 ([KL: Thm.3.18]). Π *is computationally secret (i.e. has indistinguishable encryptions in the presence of an eavesdropper), assuming Assumption 6.*

Proof by reduction

This proof will exemplify a basic framework of 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 (because usually cannot) prove unconditionally that a construction is secure, but rather, we assume some *lowe-level* primitive or problem is secure or hard to solve, and then prove the construction is secure *under* this assumption. In essence it is a proof by contradition.

To prove

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

we show the *contraposition* by a reduction

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

We do so by presenting an explicit REDUCTION, which transforms any efficient adversary $\mathcal{A}$ that succeeds in "breaking" the construction into annother efficient algorithm $\mathcal{A}'$ that sovles the problem that was assumed to be hard. This will demontrate a contradiction. It's often helpful to draw a redcution diagram. [7]

[7] **FS NOTE**: Draw Generic reduction Diagram

A few words on reductions. You should have worked with reductions in 311/350. The general purpose of a reduction is seen from two complementary perspectives:

1. Algorithmic: solving a new problem by transforming it to another problem of which an algorithm is known. [8]
2. Complexity-theoretical: related the hardness of problems, i.e., $A \leq B$ shows that Problem B is at least as hard as A. E.g., proving NP-complete problems.

[8] A mathematician interviews to be a firefighter. A dumpster vs. a table beside the dumpster;

Proof idea. Consider the indistinguihsability game. We know that in OTP, the advantage is exactly 1/2. In PRG-OTP, the only change is using a pseudorandom string as the pad. Suppose it is insecure, i.e., there is an adversary with advantage $1/2 + \varepsilon$ (ε noticeable). The only possibility to gain additional ε advantage can only come from somehow telling apart PR key from TR key. : as an extreme case, I (playing the adversary) always succeed. Namely I can come up with two messages, and figure out which one you (playing the challenger) randomly chose to encrypt. Can you take advantage of me and distinguish PR from TR? say, someone else hands you a string being either TR or PR, what can you do? Well, use it to encrypt one of the message that you choose at random, and if it were TR, you know for sure I can be correct with 1/2 chance since it is OTP. But if it PR, you know (as I promised you) I can win all the time. What'es this mean? By observing whether I won, you can tell whether the string is PR or TR.