0368.4162 Foundations of Cryptography November 7, 2019

Lecture 2: Computational Security

Lecturer: Nir Bitansky

1 Previously on Foundations of Crypto

In the previous lecture, we discussed the most basic cryptographic problem — encryption. We've defined perfect encryption and have learned that in reality it is too good to be true. We saw that if we do not want the key size n to grow with the amount of encrypted information then we have to significantly relax security:

- We have to allow a small probability that the system would break.
- We have to consider adversaries that run in bounded time.

Specifically, if the key is even slightly shorter than the message, we saw an efficient attack that simply guesses the key, and thus breaks the system with probability 2^{-n}. We also saw a $2^{n+o(n)}$-time attack that breaks the system with very high probability. In fact, the latter attack can be sped up if we allow non-determinism.

Claim 1.1. *If $P = NP$, for any cipher for messages of length ℓ, with keys of length $n < \ell - 6$, there exist two messages m_0, m_1 and an* **efficient** *attacker A such that*

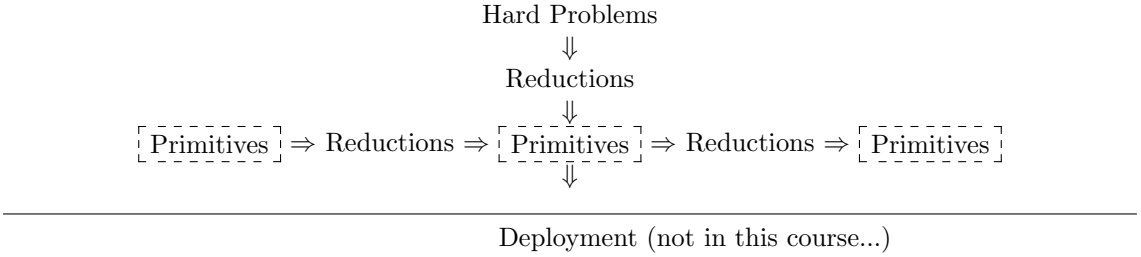
$$\Pr\left[A(ct) = b \;\middle|\; \begin{array}{l} sk \leftarrow \{0,1\}^n \\ b \leftarrow \{0,1\} \\ ct \leftarrow E_{sk}(m_b) \end{array}\right] > 0.99 \ .$$

Proof sketch. Recall that we proved the existence of messages m_0, m_1 and a distinguisher A that simply checks if the ciphertext is in the set $C_0 = \{E_{sk}(m_0)\}$. This test can be easily performed in non-deterministic polynomial time (why?). □

So, computationally hard problems are necessary for cryptography. In fact, as we shall see later on, we will need much more than the assumption that P $\neq$ NP.

2 The Layers of Crypto

With the above understanding, modern cryptography can be (very roughly) divided into three basic layers:



Layer 1: Hard Problems. These are concrete problems that we assume to be hard. Such problems may come from many different areas in different flavors:

- Number theory
- Combinatorics

- Learning
- Engineered hardness (e.g., AES)

Different assumptions may have different evidence for their hardness. Assumptions also differ in the level of confidence we have in their hardness: relations to other assumed-to-be-hard problems, etc, amount of time spent on trying to break them. Indeed, the complementing area for this layer (which we won't touch in the course but is very important) is *cryptanalysis*.

Layer 2: Primitives. These are cryptographic abstractions that obtain specific well-defined properties (e.g., secret-key encryption, one-way functions, fully homomorphic encryption, etc.) Such primitives are usually constructed and proven secure by *a reduction* to a hard problem. We will also try to reduce given primitives to other (hopefully, simpler) primitives.

Layer 3: Cryptographic Systems. Deploying cryptographic schemes in the real world to guarantee security for the user. This involves many issues and is often the bottleneck in achieving actual security:

- Verifying correctness of the implementation (naive software bugs can be disastrous).
- Integration within existing systems.
- Side channel attacks (e.g., measuring the power consumption of a device).
- Human interface pitfalls (e.g., phishing, weak passwords, etc.).

We will mostly focus on Layer 2, and sometimes touch layer 1, with two main objectives in mind:

1. Reduce cryptography to the fewest and simplest primitives possible (prove "completeness theorems").
2. Be able to base those basic primitives on a variety of hard problems, preferably well-studied ones.

There are of course additional objectives, such as making our systems as efficient as possible, which could have a tradeoff with the strength of the computational assumptions that we make, but this generally wouldn't be our focus.

3 Encryption against Computationally-Bounded Adversaries

We will now define more precisely what we mean by security against computationally-bounded adversaries, staying focused on the natural problem of encryption. Throughout, our definition of the syntax and correctness of encryption schemes will remain as in the previous lecture. We focus on security.

Definition 3.1 (Computational Security, Quantified Version)**.** *For time bound $t = t(n)$ and security error bound $\varepsilon = \varepsilon(n)$, (E, D) is (t, ε)-secure for messages of size $\ell = \ell(n)$ if for (any n) and any two messages $m_0, m_1 \in \{0,1\}^\ell$, no t-time adversary A can distinguish an encryption of one from the other with advantage greater than ε:* [1]

$$\Pr\left[A(ct) = b \;\middle|\; \begin{array}{l} sk \leftarrow \{0,1\}^n \\ b \leftarrow \{0,1\} \\ ct \leftarrow E_{sk}(m_b) \end{array}\right] \leq \frac{1}{2} + \varepsilon \ .$$

Recall that for this definition to be feasible it must be that $t \leq 2^{n+o(n)}$ and $\varepsilon \geq 2^{-n}$, and ideally we'd like our encryption schemes to come as close as possible to this level of security. This is often termed *n-bit security*. Naturally, generalizing this, a system has s-bit security, for some $s(n) \leq n$ if it is (t, ε) secure for $t = \varepsilon^{-1} = 2^s$.

[1] As long as we're talking about secret-key encryption, we'll stick with the convention that the secret key is simply chosen at random (see note in previous lecture).

The Asymptotic Approach. In practice, *every bit of security matters.* Due to efficiency considerations, the length n of keys is chosen so that $2^{s(n)}$ is sufficiently above what is considered a feasible running time for today's strongest attackers, but not much more.

Dealing with concrete security parameters could be quite a hairy business. To understand the theoretical foundations of crypto, we don't have to go there. Instead, we will take an asymptotic approach with a quite liberal interpretation of *feasible* (or *efficient*) as *polynomial*, and infeasible as *super polynomial*. We will use the following terminology:

- We say that a function $f(n)$ is polynomially-bounded if for some constant c, and all $n \in \mathbb{N}$, $f(n) \leq n^c$. We sometimes denote this by $f(n) = n^{O(1)}$.
- We will say that an algorithm A is polynomial-time if its running time is polynomially-bounded (as a function of its input size).[2] (Sometimes, we will say that A is $n^{O(1)}$-time.) More generally, we will consider probabilistic poly-time (PPT) algorithms.
- We will say that a function $\mu(n) \in [0,1]$ is negligible if it decays faster than any polynomial. That is, for any constant c, there exists n_c, such that for all $n > n_c$ it holds that $\mu(n) \leq n^{-c}$. (Sometimes, we will denote this by $\mu(n) = n^{-\omega(1)}$.)

These definitions behave quite nicely and are thus easy to work with. For instance, $n^{O(1)} \cdot n^{O(1)} = n^{O(1)}$, $\left(n^{O(1)}\right)^{O(1)} = n^{O(1)}$, and $n^{O(1)} \cdot n^{-\omega(1)} = n^{-\omega(1)}$.

Pause: How to Model Polynomial-Time Computation? One aspect that we haven't fully specified is how to exactly model adversarial algorithms. Here a natural choice is as Turing machines. Indeed, given that we only care about the running time being polynomial, all reasonable choices (e.g., RAM machines) would be equivalent. One distinction that we will make is between *uniform* algorithm and *non-uniform* ones:

- By a uniform algorithm we mean a single Turing machine A that works for any input size. In particular, A has constant-size description that unlike the running time, doesn't scale with the input size.
- A non-uniform algorithm is specified by a sequence of algorithms $A = \{A_n\}_{n\in\mathbb{N}}$, one for every input length n, and the description size of A_n may (polynomially) scale with n. This for example allows modeling a realistic setting where the attacker has some auxiliary information (as part of its description) about the world (e.g., all the ciphertexts he'd seen in its lifetime). We can think w.l.o.g about any non-uniform PPT A as a family of boolean probabilistic circuits of polynomial size.[3] (By probabilistic, you can think that the circuits include special gates that output a random bit.)

From here on in this course:

- We will consider adversarial algorithms to be non-uniform, which will often simplify our analysis. This does come at some cost, but for now we will not dwell on this.
- The algorithms of any cryptographic scheme will be uniform.

We can now define an asymptotic notion of computational security. Roughly speaking, an encryption scheme is computationally secure if it is (t,ε)-secure for any polynomial $t(n) = n^{O(1)}$ and some negligible $\varepsilon = n^{-\omega(1)}$. Let's be more explicit.

[2] In fact, we have already used this terminology when we required that the algorithms of which an encryption scheme consists are efficient.

[3] Indeed, there are several equivalent ways of modeling non-uniform algorithms. See Goldreich's book, Volume 1, Chapter 1.3 .

Definition 3.2 (Computational Security, Asymptotic Version)**.** *(E, D) is computationally secure for messages of size $\ell(n)$ if for any non-uniform PPT adversary A, there exists a negligible function $\mu(n)$, such that for any $n \in \mathbb{N}$, and any two messages $m_0, m_1 \in \{0,1\}^{\ell(n)}$:*

$$\Pr\left[A(ct) = b \;\middle|\; \begin{array}{l} sk \leftarrow \{0,1\}^n \\ b \leftarrow \{0,1\} \\ ct \leftarrow E_{sk}(m_b) \end{array}\right] \leq \frac{1}{2} + \mu(n) \ .$$

4 Toward Secret Key Encryption with Short Keys

Armed with a definition, we will now start our way toward achieving it, trying to figure out what kind of hardness assumptions it requires. Where should we start? Let's start with the simplest instance of the problem that we can think of:

Can we construct a computationally-secure encryption for messages of length $n+1$ with keys of length n?

Why start from the simplest instance? Well for once, if we cannot solve the simplest instance than we shouldn't expect to solve the general instance. Furthermore, as we shall see, looking it simple cases often helps to pinpoint the core difficulty of problems. In fact, it turns out that this is exactly the case here:

Theorem 4.1. *Assume there exists a computationally secure encryption scheme (E, D) for messages of size $\ell(n) = n + 1$ (where n is the key length). Then for any polynomial $\ell'(n)$, there exists a computationally secure encryption scheme (E', D') for messages of size $\ell'(n)$.*

For now, we will give the construction, which is quite intuitive, but only a "fake analysis" to develop some intuition. Then, we will develop some concepts and tools that will allow us to complete this proof (and practically all following proofs in the course).

(Half-Fake) Proof. Given (E, D) we construct (E', D') as follows.

The New Encryption $E'_{sk}(m)$: given a key $sk \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell'}$, the algorithm works as follows:

- Let $m = m_1 \dots m_{\ell'}$, and let $sk_0 = sk$.
- For $i = 1$ to ℓ',
  - Sample a new secret key $sk_i \leftarrow \{0,1\}^n$.
  - Sample $ct_i = E_{sk_{i-1}}(sk_i, m_i)$.
- Output the ciphertext $ct = ct_1 \dots ct_{\ell'}$.

The New Decryption $D'_{sk}(ct)$: given a key $sk \in \{0,1\}^n$ and a ciphertext ct, the algorithm works as follows:

- Let $ct = ct_1 \dots ct_{\ell'}$, and let $sk_0 = sk$.
- For $i = 1$ to ℓ',
  - Compute $(sk_i, m_i) = D_{sk_{i-1}}(ct_i)$.
- Output the message $m = m_1 \dots m_{\ell'}$.

It is not hard to see that the scheme is correct and efficient. Let's turn to security.

The following part of the proof is fake, and is only for the sake of intuition and motivation!

Our fake proof will rely on the intuitions that we've already developed for perfect secrecy. There, we have seen that perfect security is equivalent to saying that ciphertext distribution D for any message m is independent of m.

Let's imagine for a second that our encryption scheme (E, D) is perfect (although it's not), and show that (E', D') also is. We will "show" that for any $m \in \{0,1\}^{\ell'}$:

$$E'_{sk}(m) = E_{sk_0}(sk_1, m_1)E_{sk_1}(sk_2, m_2)\dots E_{sk_{\ell'-1}}(sk_{\ell'}, m_{\ell'}) \equiv E_{sk_0}(0^{n+1})E_{sk_1}(0^{n+1})\dots E_{sk_{\ell'-1}}(0^{n+1})$$

We will use the (fake) perfect security of E one step at a time: Without being too formal (this is a fake proof after all), at each step i we use the fact that the current secret key sk_i is independent of all other samples, and thus we can invoke the perfect security of the underlying encryption.

$$\begin{array}{llll} E_{sk_0}(sk_1, m_1) & E_{sk_1}(sk_2, m_2) & \dots & E_{sk_{\ell'-1}}(sk_{\ell'}, m_{\ell'}) \quad \equiv \\ E_{sk_0}(0^{n+1}) & E_{sk_1}(sk_2, m_2) & \dots & E_{sk_{\ell'-1}}(sk_{\ell'}, m_{\ell'}) \quad \equiv \\ E_{sk_0}(0^{n+1}) & E_{sk_1}(0^{n+1}) & \dots & E_{sk_{\ell'-1}}(sk_{\ell'}, m_{\ell'}) \quad \equiv \\ \vdots & \vdots & \vdots & \vdots \quad \vdots \\ E_{sk_0}(0^{n+1}) & E_{sk_1}(0^{n+1}) & \dots & E_{sk_{\ell'-1}}(0^{n+1}) \end{array}$$

□

The above type of argument is generally called *a hybrid argument* — we would like to use some local guarantee on similarity of distributions to a global guarantee, by using the local guarantee many times. While the above proof is fake (in the sense that (E, D) *is not* perfectly secret), it does capture the "right" inductive intuition. We won't be able to show that each two subsequent distributions are the same, but we'll be able to show that their computationally close in some sense.

In what follows, we'll define this concept of computational closeness, which we'll heavily rely on throughout this course.

5 Computational Indistinguishability

Intuitively, we wish to generalize the fact that distributions are equally the same to the fact they're close in the eyes of computationally bounded adversaries. Before we go all the way and do that, it is useful to first generalize similarity to *statistical closeness*; namely, the case that distributions are not only close in the eyes of bounded observers, but also for unbounded ones.

Definition 5.1 (Statistical Indistinguishability)**.** *Two distributions X, Y over the same support are ε-statistically-indistinguishable if for any (unbounded) distinguisher A:*

$$\Delta_A(X, Y) := |\Pr[A(X) = 1] - \Pr[A(Y) = 1]| \leq \varepsilon \ .$$

The quantity $\Delta(X, Y) := \max_A \Delta_A(X, Y)$ is the statistical distance between the distributions.

In what sense is this the statistical distance?

Claim 5.2.

$$\Delta(X, Y) = \frac{1}{2}\|X - Y\|_1 := \frac{1}{2}\sum_x |\Pr[X = x] - \Pr[Y = x]| \ .$$

We can connect this definition to existing intuitions from previous definitions:

Claim 5.3. *For any X_0, X_1, A:*

$$\left|\Pr\left[A(x) = b \;\middle|\; \begin{array}{c} b \leftarrow \{0,1\} \\ x \leftarrow X_b \end{array}\right] - \frac{1}{2}\right| = \frac{\Delta_A(X_0, X_1)}{2} \ .$$

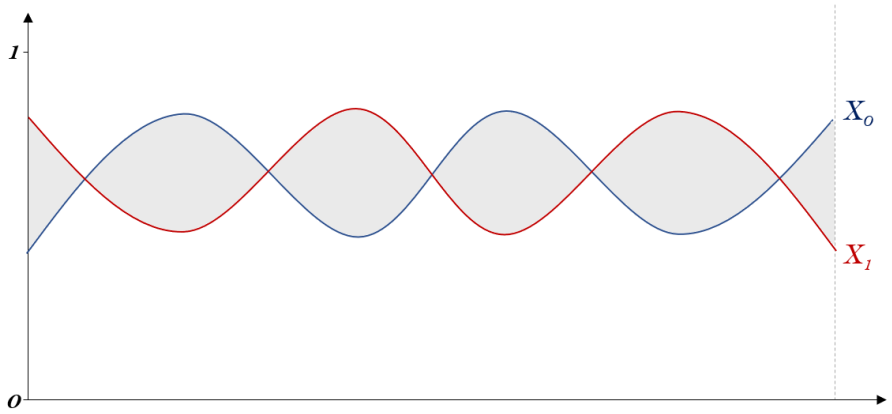


Figure 1: Illustration - The statistical distance between the two (continuous) distributions X_0, X_1 is the surface between the graphs of the two density functions.

Claim 5.4 (Triangle Inequality)**.** $\Delta_A(X,Y) \leq \Delta_A(X,Z) + \Delta_A(Z,Y)$.

We can now define our notion of computational closeness, commonly called *computational indistinguishability*, which is a natural analog of the statistical indistinguishability. Sticking with the asymptotic approach, there is no meaning to talking about computational indistinguishability (against all efficient attackers) for two specific distributions. Accordingly, we consider an asymptotic notion of *distribution ensembles*.

A distribution ensemble $X = \{X_i\}_{i\in I_n, n\in\mathbb{N}}$ is an infinite collection of distributions associated with some infinite collection $I = \uplus I_n$ of disjoint index sets. Intuitively, n represents the security parameter (size of secret key) and I_n represents some additional parameters that define the distribution.

Example: The ensemble of encryptions of length-$\ell(n)$ messages

$$\{E_{sk}(m) \mid sk \leftarrow \{0,1\}^n\}_{m\in\{0,1\}^{\ell(n)}, n\in\mathbb{N}} \ .$$

Here for every message m we have a distribution on encryptions of that message.

Definition 5.5 (Computational Indistinguishability)**.** *Two ensembles of distributions $X = \{X_i\}_{i\in I_n, n\in\mathbb{N}}$, $Y = \{Y_i\}_{i\in I_n, n\in\mathbb{N}}$, over the same set of indices $I = \uplus I_n$, are computationally indistinguishable if for any non-uniform PPT $A = \{A_n\}_{n\in\mathbb{N}}$, there exists a negligible $\mu(n)$ such that for all $n \in \mathbb{N}$ and every $i \in I_n$:*

$$\Delta_{A_n}(X_i, Y_i) := |\Pr[A_n(X_i) = 1] - \Pr[A_n(Y_i) = 1]| \leq \mu(n) \ .$$

We denote this by $X \approx_c Y$.

Claim 5.6 (Transitivity)**.** *If $X \approx_c Z$ and $Z \approx_c Y$ then $X \approx_c Y$.*

We can also think asymptotically about statistical distance and will say that two ensembles X, Y are statistically indistinguishable if the latter definition also holds for unbounded A and denote this by $X \approx_s Y$.

Proving the Security of Our Encryption. We are now ready to give a real (rather than fake) proof that our scheme E' is computationally secure if the original scheme E is computationally secure. To this we would like to prove that the encryptions of any two messages are computationally indistinguishable. Formally, we want to show that:

$$\{E'_{sk}(x) \mid sk \leftarrow \{0,1\}^n\}_{\substack{n\in\mathbb{N} \\ x,y\in\{0,1\}^{\ell'(n)}}} \approx_c \{E'_{sk}(y) \mid sk \leftarrow \{0,1\}^n\}_{\substack{n\in\mathbb{N} \\ x,y\in\{0,1\}^{\ell'(n)}}}$$

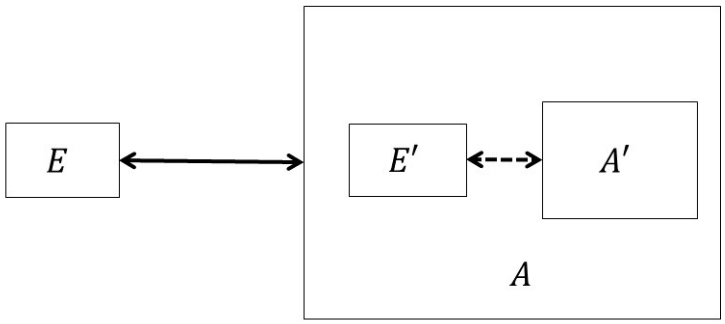


Figure 2: An efficient A' against E' can be translated into an efficient A against E; A translates an "E-type" ciphertext into an "E'-type" ciphertext.

You're probably still not used to this notation, so let's make sure we can make sense of it. It says that for any n.u. PPT A there exists a negligible μ such that for any $n \in \mathbb{N}, x, y \in \{0,1\}^{\ell'(n)}$:

$$\Delta_{A_n}(E'_{sk}(x), E'_{sk}(y)) \leq \mu(n) \ .$$

To prove the above, we will actually show that there exists a single distribution ensemble $S = \{S_n\}_n$ such that encryptions of different messages are all computationally indistinguishable from this distribution:

$$\{E'_{sk}(x) \mid sk \leftarrow \{0,1\}^n\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell'(n)}}} \approx_c \{S_n\}_{\substack{n\in\mathbb{N} \\ x\in\{0,1\}^{\ell'(n)}}} \ .$$

This is actually an equivalent definition for secure encryption, similar to the one we had for perfect security, and we can prove it using the fact that computational indistinguishability is transitive:

$$\{E'_{sk}(x) \mid sk \leftarrow \{0,1\}^n\}_{\substack{n\in\mathbb{N} \\ x,y\in\{0,1\}^{n+1}}} \approx_c \{S_n\}_{\substack{n\in\mathbb{N} \\ x,y\in\{0,1\}^{n+1}}} \approx_c \{E'_{sk}(y) \mid sk \leftarrow \{0,1\}^n\}_{\substack{n\in\mathbb{N} \\ x,y\in\{0,1\}^{n+1}}} \ .$$

We define $S = \{S_n\}$ as follows;

$$S_n \quad := \quad E_{sk=sk_0}(0^{n+1}) \quad E_{sk_1}(0^{n+1}) \quad \ldots \quad E_{sk_{\ell'-1}}(0^{n+1})$$

The Security Reduction. To prove that $S \approx_c \{E'_{sk}(x)\}_{n,x}$, we will show that any efficient adversary A' against E' can be translated into an efficient adversary A against E. This is what we call *a security reduction*; it is the bread and butter of provably-secure crypto and we'll do it all the time in this course (see figure 2).

Assume toward contradiction that there is a n.u. PPT attacker $A' = \{A_n\}_n$ and polynomial $p(\cdot)$ such that, for infinitely many n, there exists $x \in \{0,1\}^{\ell'(n)}$ such that:

$$\Delta_{A'_n}(E'_{sk}(x), S_n) = |\Pr[A'_n(E'_{sk}(x)) = 1] - \Pr[A'(S_n) = 1]| \geq 1/p(n) \ .$$

We will turn it into a non-uniform PPT attacker A that (for the same infinitely many n) will break (E, D).

Fix such n and x, we define the following hybrid distributions: for every $0 \leq i \leq \ell'$:

$$H_i := E_{sk_0}(0^{n+1}), \ldots, E_{sk_{i-1}}(0^{n+1}), E_{sk_i}(sk_{i+1}, x_{i+1}), \ldots, E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'}) \ .$$

According to this definition we have:

	H_0	$=$	$E_{sk_0}(sk_1, x_1)$	$E_{sk_1}(sk_2, x_2)$	$\dots$				$E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'})$
	H_1	$=$	$E_{sk_0}(0^{n+1})$	$E_{sk_1}(sk_2, x_2)$	$\dots$				$E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'})$
	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
$i-1$	H_{i-1}	$=$	$E_{sk_0}(0^{n+1})$	$E_{sk_1}(0^{n+1})$	$\dots$	$\boldsymbol{E_{sk_{i-1}}(sk_i, x_i)}$	$E_{sk_i}(sk_{i+1}, x_{i+1})$	$\dots$	$E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'})$
i	H_i	$=$	$E_{sk_0}(0^{n+1})$	$E_{sk_1}(0^{n+1})$	$\dots$	$\boldsymbol{E_{sk_{i-1}}(0^{n+1})}$	$E_{sk_i}(sk_{i+1}, x_{i+1})$	$\dots$	$E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'})$
	$\vdots$		$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
	$H_{\ell'}$	$=$	$E_{sk_0}(0^{n+1})$	$E_{sk_1}(0^{n+1})$	$\dots$				$E_{sk_{\ell'-1}}(0^{n+1})$

Note that $H_0 = E'_{sk=sk_0}(x)$ and $H_{\ell'} = S_n$.

Claim 5.7. *There exists $i \in [\ell']$ such that $\Delta_{A'}(H_{i-1}, H_i) \geq \frac{1}{p(n)\cdot\ell'(n)}$.*

Proof.

$$\frac{1}{p(n)} \leq \Delta_{A'}(E'_{sk}(x), S_n) = \Delta_{A'}(H_0, H_{\ell'}) \leq \sum_{i\in[\ell']} \Delta_{A'}(H_{i-1}, H_i) \leq \max_{i\in[\ell']} \Delta_{A'}(H_{i-1}, H_i) \cdot \ell'(n) \ ,$$

where the first inequality follows from the definition of A' and the second inequality from the triangle inequality. □

Notice that H_{i-1} and H_i only differ on the ith encryption:

- In H_{i-1}, we have $E_{sk_{i-1}}(sk_i, x_i)$,
- In H_i, we have $E_{sk_{i-1}}(0^{n+1})$.

We now need to construct an adversary A that breaks E. More precisely, we will show that for a random $sk_i \leftarrow \{0,1\}^n$

$$\Delta_A((E_{sk}(sk_i, x_i), sk_i), (E_{sk}(0^{n+1}), sk_i)) \geq \frac{1}{\ell'(n)\cdot p(n)} \ .$$

(Make sure you understand why this is enough.)

Adversary A: Given a ciphertext ct, and sk_i, A will sample by himself all the ciphertexts, except for the ith one, and plant ct as the ith cipher, in the following way:

$$\underbrace{E_{sk_0}(0^{n+1})}_{ct_1} \quad \underbrace{E_{sk_1}(0^{n+1})}_{ct_2} \quad \dots \quad \underbrace{ct}_{ct_i} \quad \underbrace{E_{sk_i}(sk_{i+1}, x_{i+1})}_{ct_{i+1}} \quad \dots \quad \underbrace{E_{sk_{\ell'-1}}(sk_{\ell'}, x_{\ell'})}_{ct_{\ell'}}$$

It will then run A' on this sample and output the same bit.

Note that if $ct \leftarrow E_{sk}(sk_i, x_i)$ then the sample is distributed exactly like H_{i-1} and if $ct \leftarrow E_{sk}(0^{n+1})$ then it is distributed exactly like H_i. Accordingly, we have:

$$\Delta_A((E_{sk}(sk_i, x_i), sk_i), (E_{sk}(0^{n+1}), sk_i)) = \Delta_{A'}(H_{i-1}, H_i) \geq \frac{1}{\ell'(n)\cdot p(n)} \ .$$

Crucially, in both hybrids $ct_1, \dots, ct_{i-1}, ct_{i+1}, \dots, ct_{\ell'}$ are completely independent of sk_{i-1} and could be generated efficiently. (Notice that to sample ct_{i+1}, we use the input sk_i.) □

Note on Using Non-Uniformity in the Reduction. In the above reduction, we've implicitly relied on non-uniformity. That is, even if A' was uniform, the A that we constructed was non-uniform — where does A get the value i for which A' has an advantage? non-uniformity! This use of non-uniformity is not really inherent. You can show for instance that if A chooses i at random, the reduction would still work. Alternatively, you can show that A can efficiently (and uniformly) find i, by sampling, and estimating the advantage for each i.