NYU COMPUTER SCIENCE FALL 2026

Introduction to Cryptography

CSCI-GA.3210-001 Instructor: Nir Bitansky

Lecture 2: Computational Security

1 Previously on Introduction to Crypto

In the previous lecture, we discussed the most basic cryptographic problem — encryption. We defined perfect encryption and 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 will be broken.
- 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 nondeterminism.

Claim 1.1. *If $P = NP$, then 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*

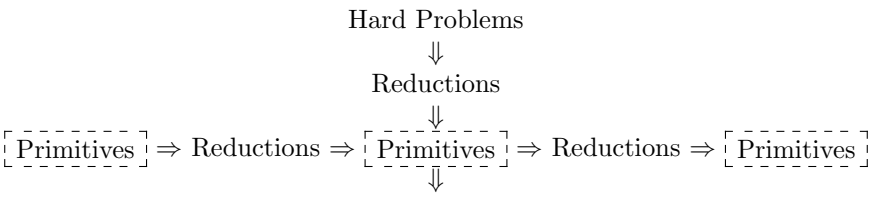
$$\Pr\left[A(ct) = b \;\middle|\; \begin{array}{l} sk \leftarrow \{0,1\}^n \\ b \leftarrow \{0,1\} \\ ct \leftarrow E_{sk}(m_b) \end{array}\right] > 0.99 \ .$$

Proof sketch. Recall that we proved the existence of messages m_0, m_1 and a distinguisher A that simply checks whether the ciphertext is in the set $C_0 = \{E_{sk}(m_0)\}$. This test can easily be performed in nondeterministic 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 ≠ 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 hardness assumptions may be supported by different kinds of evidence. Our confidence in these assumptions may depend on their relations to other problems assumed to be hard and on the amount of time spent trying to break them. Indeed, the complementary area for this layer (which we won't cover in the course but is very important) is *cryptanalysis*.

Layer 2: Primitives. These are cryptographic abstractions with 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. This layer involves deploying cryptographic schemes in the real world to guarantee security for the user. It 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 on 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. This could involve a tradeoff with the strength of the computational assumptions that we make, but it generally won'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 definitions 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 a time bound $t = t(n)$ and a 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]

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

[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 the note in the previous lecture).

Recall that for this definition to be feasible, it must be that $t \leq 2^{n+o(n)}$ and $\varepsilon \geq 2^{-n}$. 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*. More generally, a system has s-bit security for some $s(n) \leq n$ if it is (t, ε)-secure for $t = \varepsilon^{-1} = 2^s$.

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 can 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 quite a liberal interpretation of *feasible* (or *efficient*) as *polynomial*, and of infeasible as *superpolynomial*. 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 polynomial-time (PPT) algorithms.
- We will say that a function $\mu(n) \in [0, 1]$ is negligible if it decays faster than any inverse 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 exactly how to model adversarial algorithms. Here, a natural choice is to use 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* algorithms 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 a constant-size description that, unlike its 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 scale polynomially with n. This, for example, allows us to model a realistic setting where the attacker has some auxiliary information (as part of its description) about the world (e.g., all the ciphertexts it has seen in its lifetime). We can think of any non-uniform PPT A, without loss of generality, as a family of probabilistic Boolean circuits of polynomial size.[3] (You can think of probabilistic circuits as including 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.

[2] In fact, we already used this terminology when we required the algorithms that make up an encryption scheme to be 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 working toward achieving it, trying to figure out what kinds 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 scheme for messages of length $n+1$ with keys of length n?

Why start from the simplest instance? Well, for one thing, if we cannot solve the simplest instance, then we shouldn't expect to solve the general instance. Furthermore, as we shall see, looking first at 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 subsequent 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 saw that perfect security is equivalent to saying that the 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') is too. 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.

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

□

The above type of argument is generally called *a hybrid argument* — we would like to turn a local guarantee about the similarity of distributions into a global guarantee by applying 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 pair of consecutive distributions is identical, but we'll be able to show that they are computationally close in some sense.

In Lecture 3, we'll define this concept of computational closeness and use it to complete the security proof.