CS 442
Introduction to Cryptography

Lecture 6: Computational Security

Instructor: Aarushi Goel

Spring 2026

Agenda

* Computational Security
* Diffie-Hellman Assumptions
* Security Parameter
* Negligible Functions.

- HW1 is due on Feb 7.

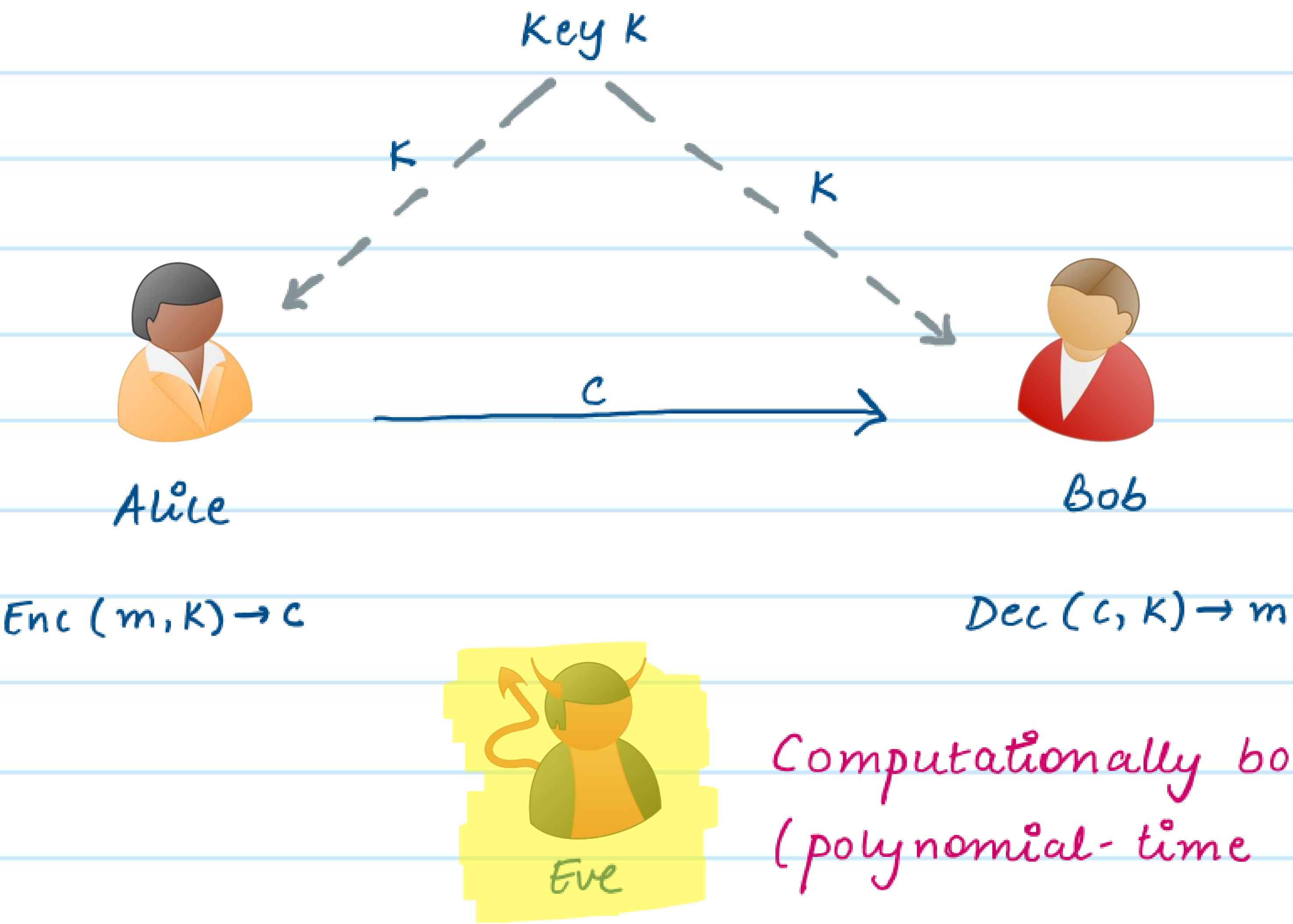
Computational Security vs Perfect Security

* For perfect secrecy, we want security against *every* Eve/Adversary.
* However, this may be an overkill.
* As we discussed earlier, there are also limitations of perfectly secure encryption schemes.
* In practice, it might be sufficient to security against computationally feasible attacks as opposed to all possible attacks

"It doesn't really matter whether attacks are impossible, only whether attacks are computationally infeasible"

* Modern cryptography is based on this principle *

Computational Security



Computationally bounded
(polynomial-time algorithm)

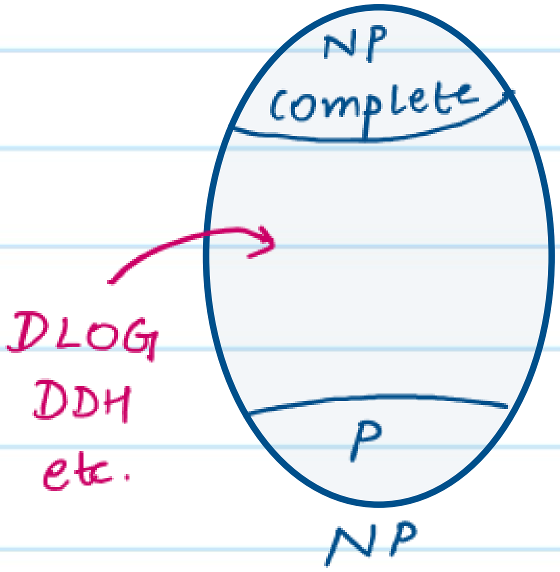
Computational Assumptions

* **Polynomial-time algorithm:** $A(x)$ with input x of length n is said to be polynomial time if A's running time is $O(n^c)$ some constant

Randomized poly-time algorithms are called probabilistic-polynomial time (PPT) algorithms.

* **NP-Problems:** Decision problems whose solution can be verified in poly time.

Example: graph isomorphism, graph 3-coloring



* **NP-complete Problems:** "hardest" problems in NP.

Is $P = NP$?

we assume $P \neq NP$.

Diffie - Hellman Assumptions

Let's sample a cyclic group $(G, \cdot)$ of order q with generator g.

* **Discrete-Log (DLOG) Assumption:** Sample $x \xleftarrow{\$} \mathbb{Z}_q$, compute $h = g^x$. Given (G, q, g, h), it is computationally hard to find x (classically)

* **Computational Diffie-Hellman (CDH) Assumption:** Sample $x, y \xleftarrow{\$} \mathbb{Z}_q$, compute $h_1 = g^x$, $h_2 = g^y$. Given (G, q, g, h_1, h_2), it is computationally hard to find g^{xy}

* **Decisional Diffie-Hellman (DDH) Assumption:** Sample $x, y, z \xleftarrow{\$} \mathbb{Z}_q$, compute $h_1 = g^x$, $h_2 = g^y$. Given (G, q, g, h_1, h_2), it is computationally hard to distinguish between g^{xy} and g^z.

* Such problems are <u>assumed</u> to be <u>computationally hard.</u>
* In other words, *<u>we don't know</u>* of any PPT algorithms to solve them

    Note that this does not mean that PPT algorithms for solving them cannot exist. It simply means that no one has been able to find one yet.

    This is why they are <u>assumed</u> to be hard

* Modern cryptographic primitives are based on the hardness assumptions of such problems. In particular, if a PPT Eve can solve these problems in poly/PPT time, then he can break security of the cryptographic scheme.

* When we talk about poly-time/PPT algorithms, what is the polynomial in? → we want polynomial in the problem size.
* For Diffie-Hellman Assumptions : Let $q = n$ bits long. then our problem size here will be $O(n)$.

Security Parameter

Or something bigger than a polynomial but smaller than exponential.

* While we don't yet have poly-time / PPT algorithms for solving these problems, they can be solved in exponential time.
* What if $n=5$? exponential in 5 may still be very small. & it may be feasible for a PPT Eve to run an algorithm that takes time exponential in 5.
* We clearly cannot base the security of our cryptographic schemes on the hardness of such small problems.
* This means, we must pick large enough problems such that exponential in "n" is still very large.

* This tunable parameter is called the security parameter

ElGammal Encryption

* KeyGen(1^λ): sample a cyclic group $(\mathbb{G}, \cdot)$ of order q with generator g.

Sample $x \xleftarrow{\$} \mathbb{Z}_q$, compute $h = g^x$.

$pk = (\mathbb{G}, q, g, h)$, $sk = x$.

$K = (pk, sk)$

In fact this part of the key can also be revealed to Eve.

Such schemes are called public-key encryption schemes. We will learn more about them later in the course.

* Enc(pk, m): $m \in \mathbb{G}$

Sample $y \xleftarrow{\$} \mathbb{Z}_q$

$c = (g^y, h^y \cdot m)$

* Dec(sk, c): Let $c = (c_1, c_2)$

$m = c_2 \cdot (c_1^{sk})^{-1}$

ElGammal Encryption

* **Correctness:** $\forall\, m,\ K \leftarrow \text{Keygen}$

$$\Pr[\text{Dec}(K, \text{Enc}(K, m)) = m] = 1$$

$$\text{Dec}(K, \text{Enc}(K,m)) = \text{Dec}(K, (g^y, h^y \cdot m)) = \text{Dec}(K, (g^y, g^{xy} \cdot m))$$
$$= g^{xy} \cdot m \cdot ((g^y)^x)^{-1} = g^{xy} \cdot m \cdot g^{-xy} = m$$

* **Security (Informal):** The ciphertext comprises of $(g^y, g^{xy} \cdot m)$, where $x, y \xleftarrow{\$} \mathbb{Z}_p$. From the DDH assumption, we know that it is computationally hard to distinguish between (g^y, g^x, g^{xy}) & (g^y, g^x, g^z), where $z \xleftarrow{\$} \mathbb{Z}_p$. This means $(g^y, g^{xy} \cdot m)$ looks similar to $(g^y, g^z \cdot m)$ to a computationally bounded Eve.

Observe that $g^z \cdot m$ is a one-time pad encryption. which we know is secure.

Security Parameter

computational

n : security parameter

→ Runtimes of algorithms and success probabilities are measured as a function of n.

* Alice, Bob run in time (fixed) polynomial in n.
* We assume Eve runs in time (arbitrary) polynomial in n.
* Eve's success probability should be a very small function in n.

* In practice, we typically set $n = 128$.

and we want the best algorithm to break the scheme to run in time $\sim 2^n = 2^{128}$.

Computationally Secure Encryption.

An encryption scheme (KeyGen, Enc, Dec) with message space $\mathcal{M}$ is computationally secure if it satisfies correctness (as defined previously) and if for every PPT Eve, the following holds in the game below.

$$\Pr[b = b'] = 1/2 + \varepsilon$$

What is ε? How do we define it?

Eve

$$\xrightarrow{m_1, m_2 \in \mathcal{M}}$$

$$\xleftarrow{c = \mathrm{Enc}(K, m_b)}$$

$$\xrightarrow{b'}$$

Challenger

$\mathrm{KeyGen} \rightarrow K$

$b \xleftarrow{\$} \{1, 2\}$

Negligible Functions

* Even the best PPT Eve should have an *extremely small* advantage
* One option is to consider exponentially small. But that is an overkill.
* We capture this using negligible functions.

Definition: A function $\nu(\cdot)$ is negligible, if for every polynomial $p(\cdot)$, we have $\lim_{n\to\infty} p(n)\cdot\nu(n) = 0$

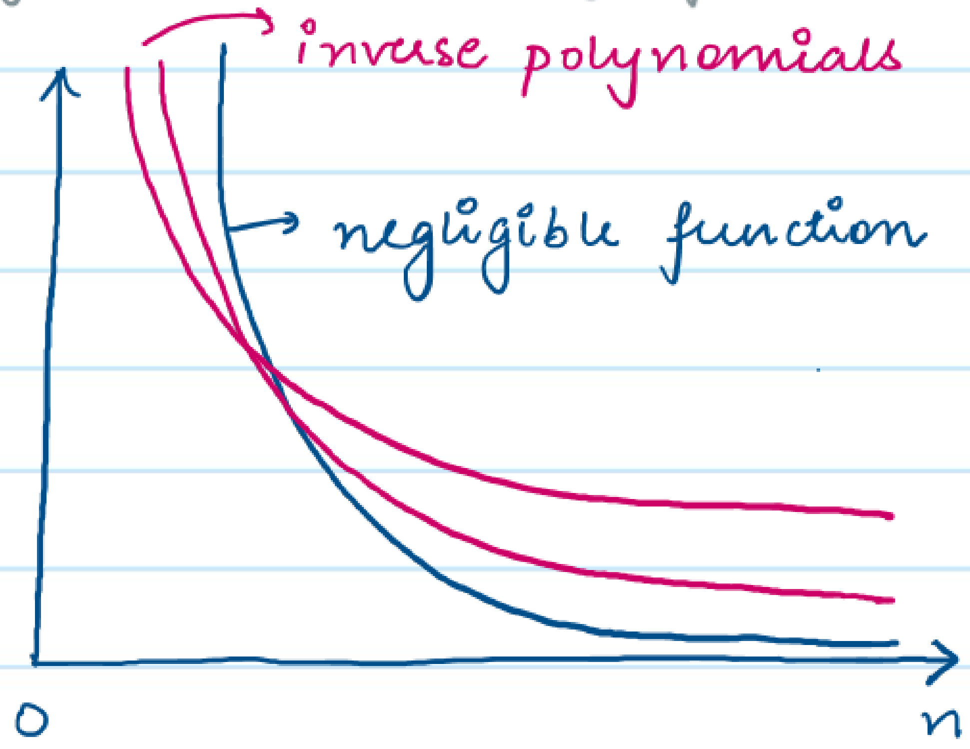
$\Rightarrow$ A negligible function decays faster than all inverse polynomial functions.

Definition: A function $\nu(n)$ is negligible if $\forall c \geq 0, \exists N$, s.t. $\forall n > N$, $\nu(n) \leq \frac{1}{n^c}$

order of quantifiers is important here (see Lecture 2)

Negligible Functions

A negligible function decays faster than all inverse polynomial functions.



Events that happen with negligible probability look to poly-time (& PPT) algorithms like they never occur

Computationally Secure Encryption.

An encryption scheme (KeyGen, Enc, Dec) with message space $\mathcal{M}$ is computationally secure if it satisfies correctness (as defined previously) and if for every PPT Eve, the following holds in the game below.

$$\Pr[b = b'] = 1/2 + \varepsilon(\lambda)$$

negligible function in the security parameter

Eve

$m_1, m_2 \in \mathcal{M}$ $\longrightarrow$

$\longleftarrow$ $c = \mathrm{Enc}(K, m_b)$

b' $\longrightarrow$

Challenger

$\mathrm{KeyGen} \rightarrow K$

$b \xleftarrow{\$} \{1, 2\}$

Examples of Negligible Functions

* **Ex 1:** $\frac{1}{2^n}$ This is negligible since for any polynomial $p(n) = n^c$, there always exists N, such that $\forall n > N$, $\frac{1}{2^n} \leq \frac{1}{n^c}$. This is because $\frac{1}{2^n}$ is exponential, so it is asymptotically smaller than any inverse polynomial $1/n^c$.

* **Ex 2:** $2^{-\omega(\log n)}$. Recall that ω is defined as follows:

$f(n) = \omega(g(n))$ if $\forall c > 0$, $\exists n_0 > 0$, s.t. $\forall n > n_0$, it holds that

$$f(n) > c \cdot g(n)$$

$$\omega(\log n) > c \cdot \log n \Rightarrow -\omega(\log n) < -c \cdot \log n$$

$$\Rightarrow 2^{-\omega(\log n)} < 2^{-c \cdot \log n}$$

$$< 2^{-\log n^c}$$

$$< \frac{1}{n^c}$$

Examples of Functions that are Not Negligible

* **Ex 1:** $\frac{1}{n^2}$ This is not negligible since for polynomial n^3, & any $n \geq 1$,

$$\frac{1}{n^2} \nleq \frac{1}{n^3}$$

* **Ex 2:** Let $f(n)$ & $g(n)$ be negligible functions.

Then $\frac{f(n)}{g(n)}$ may or may not be negligible.

- Let $f(n) = \frac{1}{2^n}$ & $g(n) = \frac{1}{4^n}$

$f(n)/g(n) = \frac{4^n}{2^n} = 2^n$ which is clearly not negligible

- Let $f(n) = \frac{1}{4^n}$ & $g(n) = \frac{1}{2^n}$

$f(n)/g(n) = \frac{1}{2^n}$ which is negligible.