CS4501 ~~Cryptographic Protocols~~
Intro to Cryptography Speedrun
Lecture 12: Bounded Adversaries OWFs, Assumptions on Groups

https://jackdoerner.net/teaching/#2026/Spring/CS4501

Introducing *Oblivious Transfer.*

- Let $\mathscr{M}$ be some message space, and $m_0, m_1 \in \mathscr{M}$ be two messages known to P_1.
- The *Oblivious Transfer* functionality allows P_2 to receive *one* message of its choice.
- P_1 isn't allowed to learn which of the two messages P_2 received.
- P_2 isn't allowed to learn anything about the message that it didn't choose.

Oblivious Transfer is Fundamental

1

May 20, 81

How to Exchange Secrets
by
Michael O. Rabin

Introduction. Bob and Alice each have a secret, S_B and S_A, respectively, which they wish to exchange. For example, S_B may be the password to a file that Alice wants to access (we shall refer to this file as Alice's file) and S_A ~~the password~~ the password to Bob's file. ~~Note that each is~~ Can they set up a protocol to exchange the secrets without using a trusted third party and without a safe mechanism for the simultaneous exchange

- OT is generally considered to be the minimal functionality that is *complete* for dishonest-majority multiparty computation.
- If something can be securely computed in the presence of a dishonest majority, then it can be securely computed assuming only OT.
- OT is a *general assumption*, meaning it can be constructed from a large variety of *specific mathematical assumptions*, but using it does not bind you to any specific mathematical assumption. It gives us a *modular foundation* for MPC.
- First described by Michael O. Rabin in 1981 in his (hand written!) paper “How to Exchange Secrets”.
- This is arguably the first MPC paper.

Oblivious Transfer is Fundamental

Founding Cryptography on Oblivious Transfer

Joe Kilian*
MIT

Abstract

Suppose your netmail is being erratically censored by Captain Yossarian. Whenever you send a message, he censors each bit of the message with probability $\frac{1}{2}$, replacing each censored bit by some reserved character. Well versed in such concepts as redundancy, this is no real problem to you. The question is, can it actually be turned around and used to your advantage? We answer this question strongly in the affirmative. We show that this protocol, more commonly known as *oblivious transfer*, can be used to simulate a more sophisticated protocol, known as *oblivious circuit evaluation*([Y]). We also show that with such a communication channel, one can have completely noninteractive zero-knowledge proofs of statements in NP. These results do not use any complexity-theoretic assumptions. We can show that they have applications to a variety of models in which oblivious transfer can be done.

*Research supported in part by a Fannie and John Hertz foundation fellowship, and NSF grant 865727-CCR. Some of this research was done while working at BellCore.

1 Introduction

1.1 Eliminating assumptions: the problem at hand.

Cryptographers seldom sleep well([M]). Their careers are frequently based on very precise complexity-theoretic assumptions, which could be shattered the next morning. A polynomial time algorithm for factoring would certainly prove more crushing than any paltry fluctuation of the Dow Jones. The effect a proof that $\mathcal{P} = \mathcal{NP}$ would have is unspeakable.

More recently, cryptographers have managed to hedge their investments. There have been some exciting new results on proving completeness theorems for multi-party protocols, due to [BGW], and [CCD]. In the area of interactive proof theory, a multi-party generalization of interactive proof systems has also been proposed [BGKW] (also, [FRS]) in which anything provable can be proven in zero-knowledge.

Two-party protocol problems have experienced no such breakthrough. The multi-party protocols do not work if half the players are malicious or trying to get information. Hence, they do not apply to the two-party case. While there is a very mature theory of two-party protocols(notably, [Y] and [GMW1]),the vast majority of the work rests on various cryptographic assumptions. The best that has been done so far is to prove theorems based on more general cryptographic assumptions, such as "trapdoor functions exist," rather than specific assumptions, such as "factoring is

- The completeness of OT for malicious, dishonest-majority MPC was proven in 1988 by Kilian.
- "Cryptographers seldom sleep well [M]."

> [M] Micali, Silvio, Personal Communication.

- Cryptography is based upon *assumptions* about the nature of the world, which are not known to be true. It would be catastrophic if they are false.
- BGW leverages information theory to avoid these concerns when enough parties are honest.
- Where we're going next, we have no option but to face up to the monster in the closet.

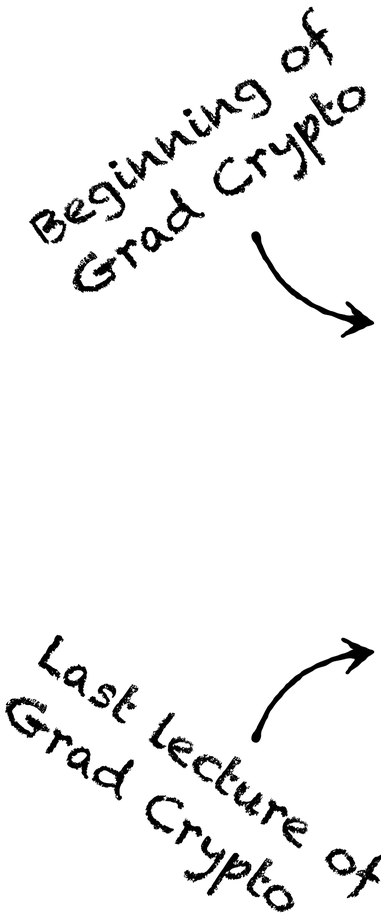
The Three Main Characters of this Class

1. Shamir Secret Sharing 
2. Oblivious Transfer 
3. Zero-Knowledge Proofs

The next few lectures will be about constructing OT.

Before we can start writing protocols, we need to understand how to reason about imperfect security in general, and develop a new toolkit.

Roadmap

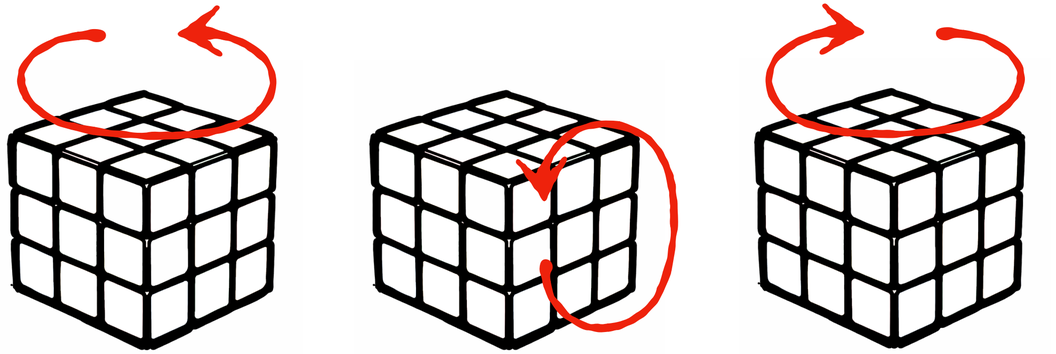


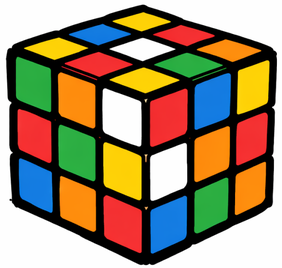
- Bounded Adversaries and Computational Security.
- Building the toolkit: One-way Functions.
- Building the toolkit: Assumptions on Groups.
- A simple application: Key Agreement.
- Ensembles and Computational Indistinguishability.
- Building the toolkit: Pseudorandomness, Symmetric Encryption, Public-key Encryption.
- Computational Security for protocols.
- Realizing Oblivious Transfer.

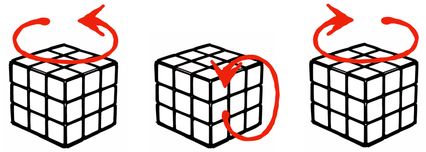
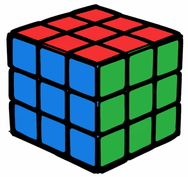
Impossible vs Very Hard

- So far, our ideas about security have been rooted in *perfect indistinguishability*. It's intuitive to conclude that no distinguisher can tell two distributions apart when they are *identical*.
- Since our notion of security had nothing to do with the computational power, we allowed adversaries and distinguishers to have *unbounded* computational power. Our adversaries and distinguishers could do things like factor products of two large primes, or solve NP-hard problems, and it didn't help them break BGW.
- Now we want to reason about *computationally bounded adversaries*. We will return to *distinguishers* later: for now, we will consider *security games* in which the adversary must solve a puzzle that we have set up for it in a limited amount of time.
- Our first puzzle is defined by some function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of finite size. We will give the adversary $y \in \mathcal{Y}$, and it wins the game if it can find $x \in \mathcal{X}$ such that $f(x) = y$ before the time is up. We don't want the adversary to win.

A Bad Analogy

$x =$  $\cdots$

$y =$ 

$f(x)$ applies  to  yielding 

Bounded Adversaries

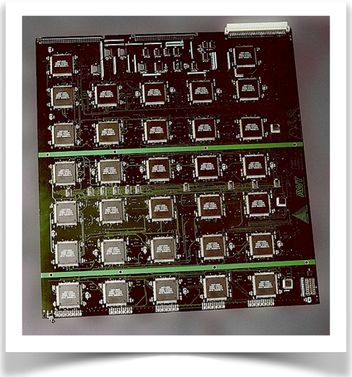
- If we wanted to achieve something like *perfect* security for the kind of puzzle we just described, we might insist that for every $\mathcal{A}$ there exists some $y \in \mathcal{Y}$ such that $\Pr[f(\mathcal{A}(y)) = y] = 0$. *Is it actually possible to achieve this?*
- *It isn't!* We have to allow some probability that $\mathcal{A}$ succeeds, because it can always just guess a solution! There are finitely many possibilities, and thus a nonzero probability that it succeeds. The *best* we can do is make the probability "very small."
- This means there are two things we need to quantify: the **runtime** of $\mathcal{A}$, and the **probability** that it solves the puzzle and wins the game.
- If we can prove that $\mathcal{A}$ has sufficiently-small (to be defined) probability of solving the puzzle, even if it runs for as much time as we can reasonably expect it to in the real world (also to be defined), then we say the puzzle is *computationally secure.*

The Concrete Approach

- The simplest way to apply this intuition is to say that our puzzle is "(t, ε)-*secure*" if every adversary that runs for time $\leq t$ has probability $\leq \varepsilon$ of winning the game.
- For example, $t = 2^{60}$ seconds is longer than the age of the universe; $\varepsilon = 2^{-20}$ is less likely than being struck by lightning in a year.
- This kind of reasoning is important in practice because it helps us to set parameters, but it's tricky to prove or generalize, and it might be sensitive to hardware details.
- Even more concerningly, this kind of analysis doesn't necessarily tell us anything about adversaries that run for time (e.g.) $2t$, $t + 1$, or $t/2$!
- Nevertheless, this is the kind of guarantee that we assume is provided by practical ciphers like AES, and practical hash functions like SHA2, which have fixed circuit descriptions. Note that even this is an *assumption*. We often find out it is wrong!

Pitfalls of the Concrete Approach

- The DES cipher was standardized in 1977, and broken by brute force in 1997. A brute force attack is one in which you simply enumerate every possible input. This attack succeeded not because somebody found a flaw, but because computers got more powerful.
- The first time anyone broke DES in practice it took 96 days. One year later, the Electronic Frontier Foundation built a machine that could break DES in two days, but it cost $250000.



DES cracker machine

- In 2012, the creator of Signal messenger was operating an online service to break DES for $20.
- Similarly, the SHA1 hash function was standardized in 1995, and broken in 2017 through a combination of cryptanalysis and brute force. By 2020, the cost of finding a collision was estimated to be about $11000. SHA1 is still widely deployed!

Toward an *Asymptotic* Notion of Security

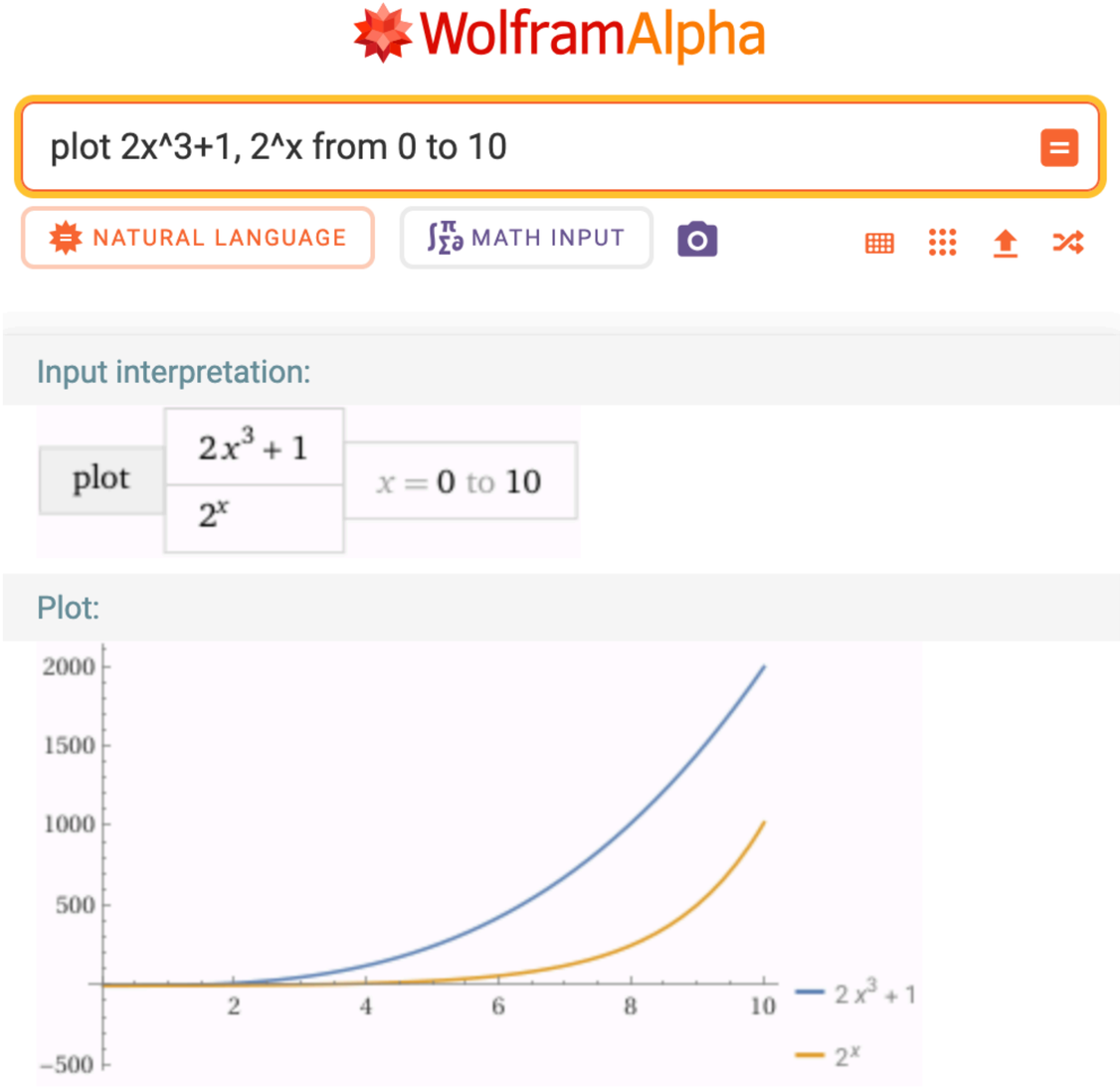
- We will make a simple but plausible assumption about the world: the adversary's computational power grows over time at a rate that's not too much faster than the honest parties'. Formally, we will assume the adversary and the honest parties are in the same *complexity class*.
- We will design our schemes with a parameter that adjusts the difficulty of winning a security game. It will usually be the case that the cost for an honest party to use a cryptographic scheme goes up with the difficulty of breaking it.
- As the computation power of honest parties grows, they can afford to raise the difficulty. As the computation power of adversaries grows, they can break a fixed difficulty level faster.
- Our goal is to make sure that, relative to an increase in honest computation power, the difficulty of the scheme goes up *asymptotically faster* that the adversary's power does. In other words: we want to design cryptography that becomes *more secure* (i.e. *harder to break*) as computers become more powerful, not less!

Review: Complexity Classes

- By now you are all hopefully comfortable with the notion of a polynomial. When discussing secret sharing, it was important that we were strict about our language, and when we said "f is a polynomial," we meant that it has the form $f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_d \cdot x^d$, where $a_0, \ldots, a_d$ are constants.
- When discussing complexity classes, we will talk about functions that are *dominated by polynomials*. A function f is dominated by a polynomial if there exists a pair of constants d, x_0 such that $\forall x \geq x_0$, it holds that $f(x) \leq x^d$.
- We will often call such functions *polynomial* even if they aren't (e.g. $x^5 \cdot \log(x)$).
- Similarly, a function f is *dominated by exponentials* (or simply *an exponential function*) if there exists some constants c, x_0 and some polynomial p such that $\forall x \geq x_0$, it holds that $f(x) \leq c^{p(x)}$.
- For example: 2^x, 2^{x^2}, 100000^x, but *not* 2^{2^x}.
- **Question:** *how do these definitions relate to what you might have seen in calculus?*

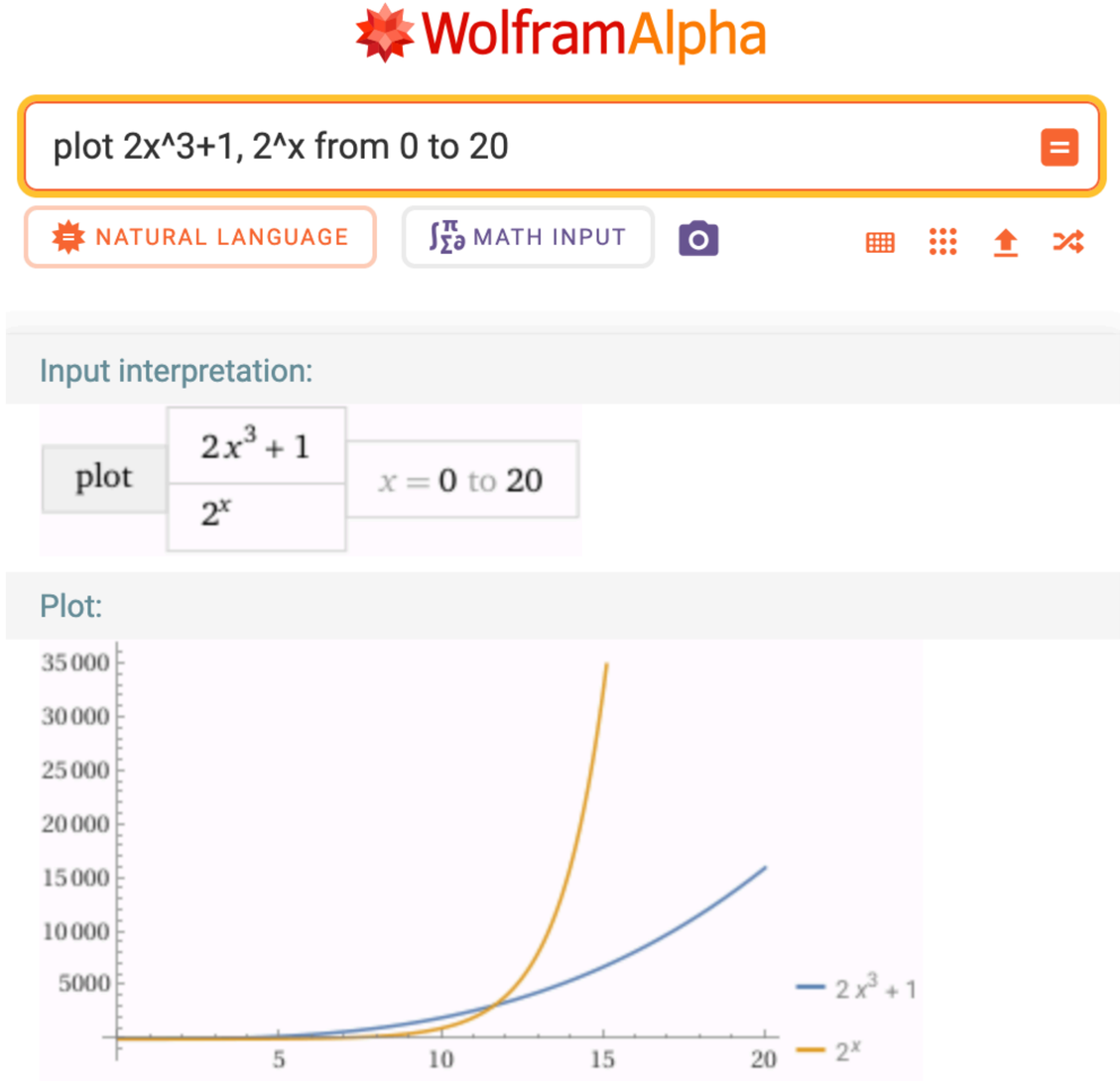
Review: Complexity Classes

- If the a function g dominates a function f, we cannot necessarily conclude that $f(x) \leq g(x)$ for every value of x.



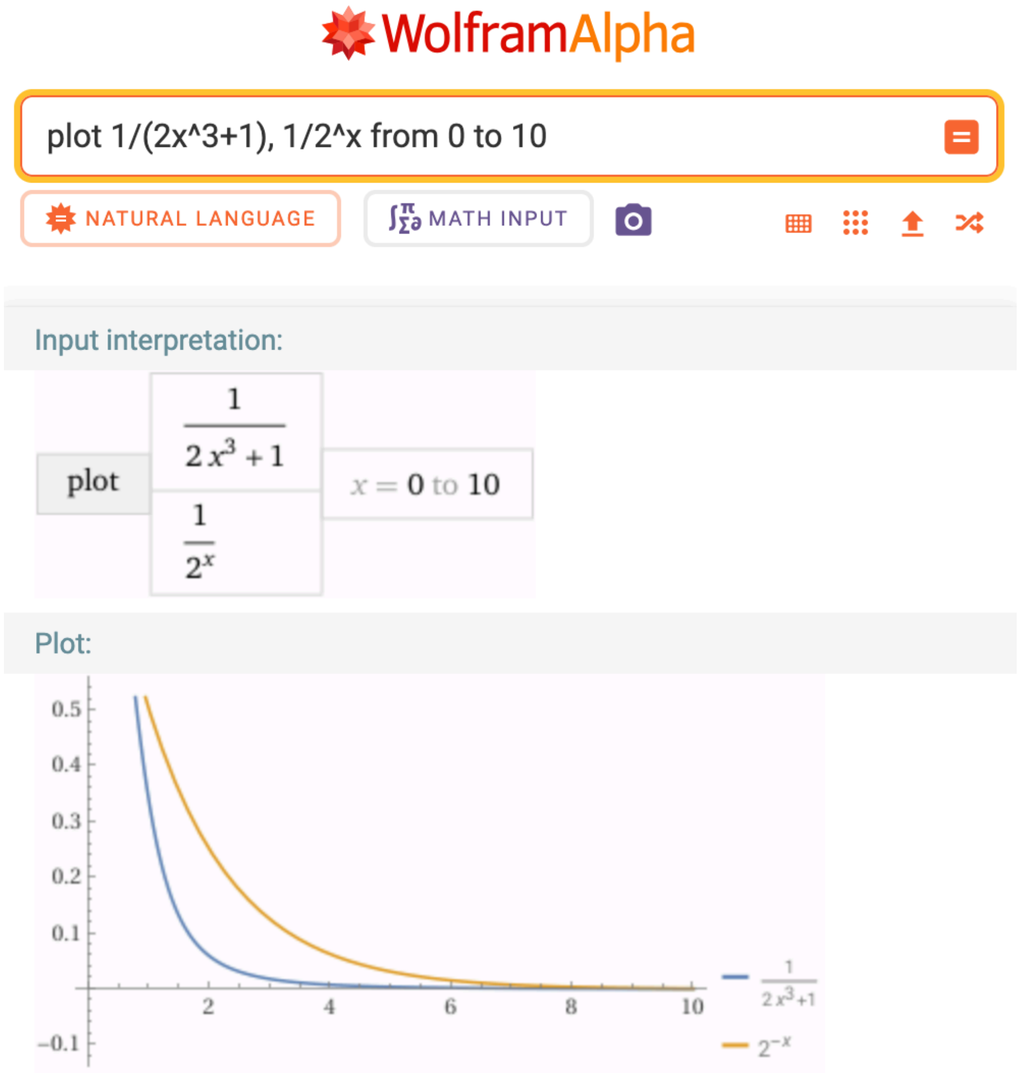
Review: Complexity Classes

- If the a function g dominates a function f, we cannot necessarily conclude that $f(x) \leq g(x)$ for every value of x.
- What we can conclude is that g *grows faster* and there will be some fixed value x_0 above which g overtakes f permanently.
- Logarithmic functions are dominated by polynomials, which are in turn dominated by exponentials, which are dominated by double exponentials, etc.
- Functions that are *not* dominated by polynomials, are called *superpolynomial*. Such functions might not be exponential!



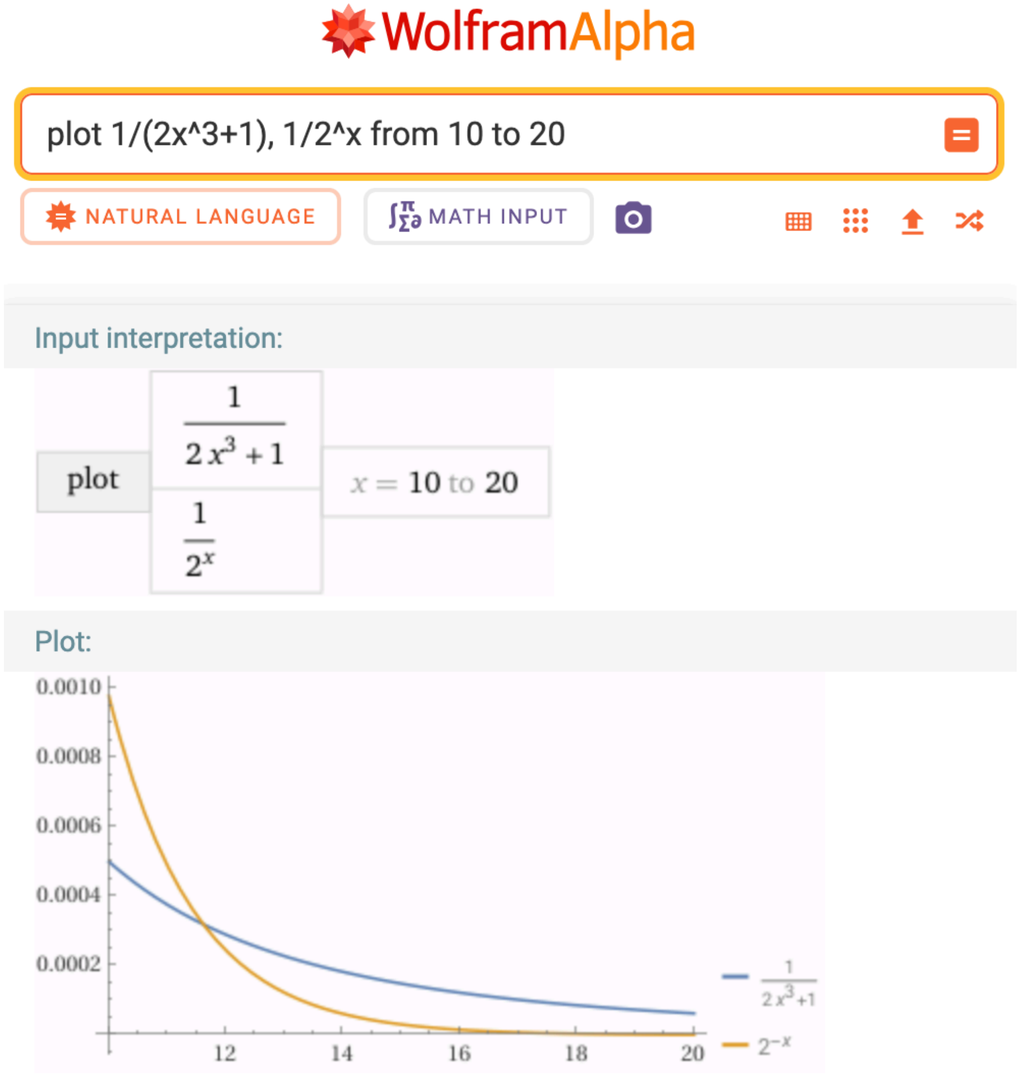
Review: Diminishing Functions

- These function classes also have inverses, and we can compare the rates at which they approach zero.
- Similar to the last example, an inverse exponential function might be greater than an inverse polynomial function at some particular input.



Review: Diminishing Functions

- These function classes also have inverses, and we can compare the rates at which they approach zero.
- Similar to the last example, an inverse exponential function might be greater than an inverse polynomial function at some particular input.
- However, an inverse exponential function diminishes faster than any inverse polynomial, so it will be smaller for all *big enough* inputs.



What "Very Small" Means

- Looking ahead, we will allow the adversary's computing power to grow polynomially. We want the probability that it wins a security game to diminish *faster than that.* This leads us directly to the following definitions:

Definition 1 (negligible fn): A function ε is *negligible* if $\forall c \in \mathbb{N}\ \exists x_0 \text{ s.t. } \varepsilon(x) < x^{-c}$. In other words, a negligible function diminishes faster than *any* inverse polynomial.

Definition 2 (non-negligible fn): A function δ is *non-negligible* if it is not negligible. That is, $\exists c \in \mathbb{N} \text{ s.t. } \forall x_0 \in \mathbb{N}\ \exists x \geq x_0 \text{ s.t. } \delta(x) \geq x^{-c}$. Or, to put it another way, there is some polynomial p such that for *infinitely many* values of x, $\delta(x) \geq 1/p(x)$.

- Notice that a non-negligible function might not be inverse polynomial! Consider the function $f(x)$ that outputs x if x is even, or 2^{-x} if x is odd. Thus it will be convenient to name the class of functions that dominate an inverse polynomial:

Definition 3 (noticeable fn): A function γ is *noticeable* if $\exists c, x_0 \in \mathbb{N} \text{ s.t. } \forall x \geq x_0\ \gamma(x) \geq x^{-c}$.

The Asymptotic Approach

- We will introduce a single variable called the *security parameter* (usually denoted n, λ, or κ, but we will use κ) and define both the runtime and the success probability of the adversary to be functions of this variable.

Definition 4 (PPT): An algorithm A runs in *probabilistic polynomial time* (PPT) if there exists a polynomial p such that for every input $x \in \{0,1\}^*$ and every random tape $r \in \{0,1\}^*$, the computation of $A(x; r)$ halts within $p(|x|)$ steps.

- In the context of theory, when we say something is "efficient", we usually mean PPT.
- Since the time complexity of an algorithm is judged relative to the *length* of its input, we will encode κ in unary (i.e. as a string of κ ones). We write this as 1^κ. We will *not* count messages received as "input" for the sake of runtime bounds.
- Recall our example: we have some function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of finite size. We will give the adversary $y \in \mathcal{Y}$, and it wins the game if it can find $x \in \mathcal{X}$ such that $f(x) = y$ before the time is up.

The Asymptotic Approach

- Since the time complexity of an algorithm is judged relative to the *length* of its input, we will encode κ in unary (i.e. as a string of κ ones). We write this as 1^κ. We will *not* count messages received as "input" for the sake of runtime bounds.
- Recall our example: we have some function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are sets of finite size. We will give the adversary $y \in \mathcal{Y}$, and it wins the game if it can find $x \in \mathcal{X}$ such that $f(x) = y$ before the time is up.
- We need the difficulty of the puzzle to scale with κ. For now, let's say that for every value of κ we have a domain $\mathcal{X}_\kappa$ and range $\mathcal{Y}_\kappa$ that yield the right difficulty. Each $\mathcal{X}_\kappa$ is finite, but we will say that f can take an input from any $\mathcal{X}_\kappa$ for $\kappa \in \mathbb{N}$.
- The asymptotic approach asserts something like: f is a "secure puzzle" if, for all PPT adversaries $\mathcal{A}$, there exists some negligible function ε and a puzzle instance $y_\kappa \in \mathcal{Y}_\kappa$ for every $\kappa \in \mathbb{N}$ such that $\Pr[f(x') = y_\kappa : x' \leftarrow \mathcal{A}(1^\kappa, y_\kappa)] \leq \varepsilon(\kappa)$.

Why These Choices?

- We can make other choices and end up with self-consistent outcomes, however:
- Polynomially bounded runtime matches our empirical experience about what kinds of algorithms are feasible to run. It's not feasible to run exponential-time algorithms in practice except on very small inputs. See also **Cobham's Thesis**.
- Polynomials and negligibles are well-behaved under composition:

Proposition 1: If p_1, p_2 are polynomial functions, then $p_3(x) = p_1(x) \cdot p_2(x)$ is also polynomial. Thus, running a polynomial time subroutine polynomially-many times yields a algorithm that is polynomial time over all.

Proposition 2: If $\varepsilon_1, \varepsilon_2$ are negligible functions, then $\varepsilon_3(x) = \varepsilon_1(x) + \varepsilon_2(x)$ is also negligible. Thus if the adversary can win by causing one of two events to occur, and both events occur with negligible probability, then the adversary wins with negligible probability.

Why These Choices?

Proposition 1: If p_1, p_2 are polynomial functions, then $p_3(x) = p_1(x) \cdot p_2(x)$ is also polynomial. Thus, running a polynomial time subroutine polynomially-many times yields a algorithm that is polynomial time over all.

Proposition 2: If $\varepsilon_1, \varepsilon_2$ are negligible functions, then $\varepsilon_3(x) = \varepsilon_1(x) + \varepsilon_2(x)$ is also negligible. Thus if the adversary can win by causing one of two events to occur, and both events occur with negligible probability, then the adversary wins with negligible probability.

Proof: Let p be some polynomial and let $q(x) = 2p(x)$.

By the definition of negligible functions, $\exists x_1 \in \mathbb{N} \text{ s.t. } \forall x \geq x_1,\ \varepsilon_1(x) < 1/q(x)$

Similarly, $\exists x_2 \in \mathbb{N} \text{ s.t. } \forall x \geq x_2,\ \varepsilon_2(x) < 1/q(x)$

If we let $x_3 := \max(x_1, x_2)$ then $\forall x \geq x_3,\ \varepsilon_3(x) = \varepsilon_1(x) + \varepsilon_2(x) < \dfrac{2}{q(x)} = \dfrac{1}{p(x)}$

Since this holds for *every* polynomial p, ε_3 is a negligible function. ∎

Why These Choices?

Proposition 2: If $\varepsilon_1, \varepsilon_2$ are negligible functions, then $\varepsilon_3(x) = \varepsilon_1(x) + \varepsilon_2(x)$ is also negligible. Thus if the adversary can win by causing one of two events to occur, and both events occur with negligible probability, then the adversary wins with negligible probability.

Proof: Let p be some polynomial and let $q(x) = 2p(x)$.

By the definition of negligible functions, $\exists x_1 \in \mathbb{N} \text{ s.t. } \forall x \geq x_1,\ \varepsilon_1(x) < 1/q(x)$

Similarly, $\exists x_2 \in \mathbb{N} \text{ s.t. } \forall x \geq x_2,\ \varepsilon_2(x) < 1/q(x)$

If we let $x_3 := \max(x_1, x_2)$ then $\forall x \geq x_3,\ \varepsilon_3(x) = \varepsilon_1(x) + \varepsilon_2(x) < \dfrac{2}{q(x)} = \dfrac{1}{p(x)}$

Since this holds for *every* polynomial p, ε_3 is a negligible function. ∎

Proposition 3: If p is a polynomial function and ε_1 is a negligible function, then $\varepsilon_2(x) = p(x) \cdot \varepsilon_1(x)$ is also a negligible function.

Proof: Homework?

Consider the following Scenario

- Suppose we have a cryptographic scheme that has requires computation time κ^2 for honest parties, and achieves security against all PPT adversaries, except with probability $2^{-\kappa}$ that the adversary breaks the scheme.
- If we assume honest parties have a time budget of t, then they can set $\kappa = \sqrt{t}$.
- The adversary's runtime is bounded by some arbitrary polynomial in κ. This polynomial could be *much larger* than κ^2. For concreteness, let us assume it is κ^{100}. Initially, the adversary can work for t^{50} time to win with probability $2^{-\sqrt{t}}$.
- Over time, computers become more powerful. Now honest parties have a time budget of $4t$, which means they can set $\kappa = 2\sqrt{t}$.
- On the other hand, the adversary's time budget grows to $2^{100}t^{50}$. If the adversary uses every bit of its new time budget, it can break the scheme with probability $2^{-2\sqrt{t}} = 2^{-\sqrt{t}}/4$.

Consider the following Scenario

- If we assume honest parties have a time budget of t, then they can set $\kappa = \sqrt{t}$.
- The adversary's runtime is bounded by some arbitrary polynomial in κ. This polynomial could be *much larger* than κ^2. For concreteness, let us assume it is κ^{100}. Initially, the adversary can work for t^{50} time to win with probability $2^{-\sqrt{t}}$.
- Over time, computers become more powerful. Now honest parties have a time budget of $4t$, which means they can set $\kappa = 2\sqrt{t}$.
- On the other hand, the adversary's time budget grows to $2^{100}t^{50}$. If the adversary uses every bit of its new time budget, it can break the scheme with probability $2^{-2\sqrt{t}} = 2^{-\sqrt{t}}/4$.
- The honest party made the security parameter twice as big and had to work four times harder, but the adversary had to work 2^{100} times harder to achieve $1/4$ of its original success probability!

Toward One-way Functions

- Recall again our example: f is "secure puzzle" if, for all PPT adversaries $\mathcal{A}$, there exists some negligible function ε and a puzzle instance $y_\kappa \in \mathcal{Y}_\kappa$ for every $\kappa \in \mathbb{N}$ such that $\Pr[f(x') = y_\kappa : x' \leftarrow \mathcal{A}(1^\kappa, y_\kappa)] \leq \varepsilon(\kappa)$.
- This still isn't quite right. There are two problems that would make this kind of object hard to use as-is. *Can you see them?*
  1. We don't know whether it's possible to find a *hard instance* y_κ efficiently. It might be the case that almost all values in $\mathcal{Y}_\kappa$ are easy to invert!
  2. We don't know whether it's possible to evaluate f efficiently.
- Let's fix those problems, and while we're at it, we'll define $\mathcal{X}_\kappa = \{0,1\}^\kappa$.

Definition 5: The One-Way Function Game

- The *one-way function game* $\mathsf{OWFGame}_{f,\mathcal{A}}(\kappa)$ is played between a "challenger" and the adversary. The output of the game is the output of the challenger. The game is as follows:

$x \leftarrow \{0,1\}^\kappa$

$y := f(x)$

$1^\kappa, y$

$x' \in \{0,1\}^\kappa$

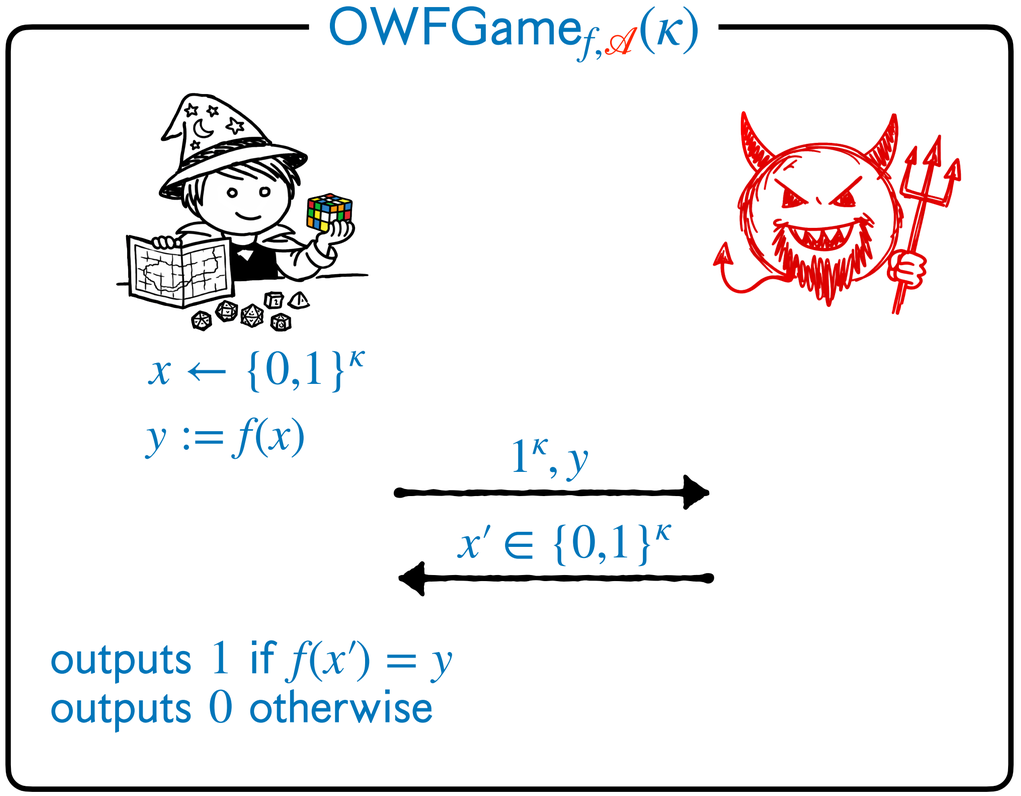
outputs 1 if $f(x') = y$
outputs 0 otherwise

One-Way Functions

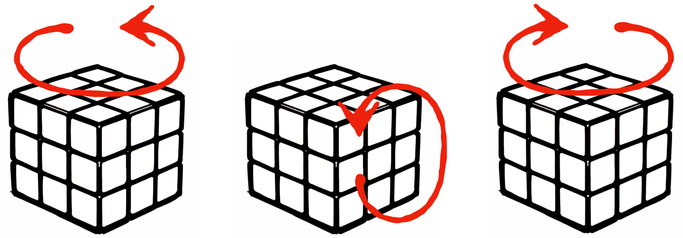
Definition 5 (OWF): $f : \{0,1\}^* \to \{0,1\}^*$ is a *one-way function* (OWF) if:

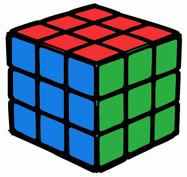
1. f is PPT.
2. $\forall$ PPT $\mathcal{A}$ $\exists$ a negligible function ε such that $\forall \kappa \in \mathbb{N}$, $\Pr[\mathsf{OWFGame}_{f,\mathcal{A}}(\kappa) = 1] \leq \varepsilon(\kappa)$

- Intuitively, f is easy to compute, but very hard to invert.
- Importantly, this holds for *average-case* (i.e. random) inputs.

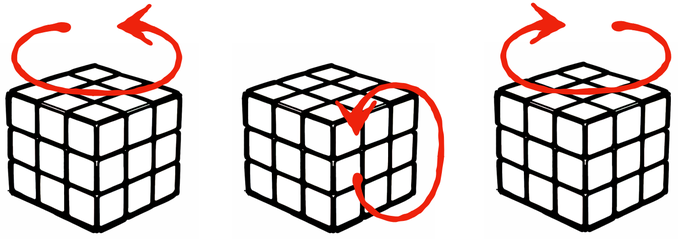
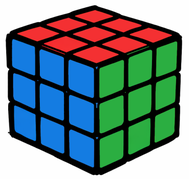


A Continuingly Bad Analogy

It's very easy to sample a random $x =$ 

and apply it to  in order to get $y =$  .

It's much harder to start from $y =$  and work out an

$x' =$  that you can do in reverse to get  .

It Will be Useful to Define a Slight Variation

Definition 6 (OWF Family): a set of functions $\{f_i : \mathcal{X}_i \to \mathcal{Y}_i\}_{i \in I}$ indexed by some set I is a *one-way function family* if:

1. There exists a PPT algorithm $\mathsf{OWFGen}(1^\kappa)$ that outputs some $i \in I$.
2. $\forall i \in I, f_i$ is PPT.
3. $\forall$ PPT $\mathcal{A}$ $\exists$ a negligible ε such that $\forall \kappa \in \mathbb{N}$, $\Pr\left[\mathsf{OWFFGame}_{\mathsf{OWFGen},\mathcal{A}}(\kappa) = 1\right] \leq \varepsilon(\kappa)$

- The main difference from Definition 5 is that the function's description depends upon κ, and it might be randomized.

$\mathsf{OWFFGame}_{\mathsf{OWFGen},\mathcal{A}}(\kappa)$

$i \leftarrow \mathsf{OWFGen}(1^\kappa)$

$x \leftarrow \mathcal{X}_i$

$y := f_i(x)$

$1^\kappa, i, y$ →

← $x' \in \mathcal{X}_i$

outputs 1 if $f_i(x') = y$
outputs 0 otherwise

OWFs are the most fundamental primitive in cryptography. They're the foundation for almost everything else.

We don't know if OWFs *actually exist!*
To construct them we need to make assumptions that have never been proven!

This is why Micali rarely sleeps well: It might turn out that cryptography (as we usually mean it) is impossible!

If OWFs *do* exist, they have important implications: for example, we can easily prove that $P \neq NP$ using OWFs.

Do you believe in One-way Functions?

If you sometimes type sensitive data into your web browser and expect it not to leak, then you *must* believe in OWFs!

There are many kinds assumptions
that enable us to construct OWFs.

Since we already learned about Groups,
we'll focus on assumptions about those.

We need a bit more Group Theory...

Review (Lecture 5): Groups

Let $\mathbb{G}$ be a set and $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ be a binary operation such that $\mathbb{G}$ is closed under $\star$.

Definition 7: $(\mathbb{G}, \star)$ is a *group* if and only if all of the following axioms hold:

1. Associativity: $\forall a, b, c \in \mathbb{G},\ a \star (b \star c) = (a \star b) \star c$.
2. Identity: there exists an identity element i such that $\forall a \in \mathbb{G}$ we have $i \star a = a \star i = a$.
3. Inverses: $\forall a \in \mathbb{G}\ \exists b \in \mathbb{G}$ such that $a \star b = i$.

Definition 8: $(\mathbb{G}, \star)$ is a *commutative (a.k.a. abelian) group* if it is a group, and:

4. Commutativity: $\forall a, b \in \mathbb{G}$ we have $a \star b = b \star a$.

Definition 9: the *order* of $(\mathbb{G}, \star)$ is the size of $\mathbb{G}$.

Review (Lecture 5): Finite Groups by Example

Consider $(\mathbb{Z}_m, +)$ where $+$ is interpreted as addition modulo m.

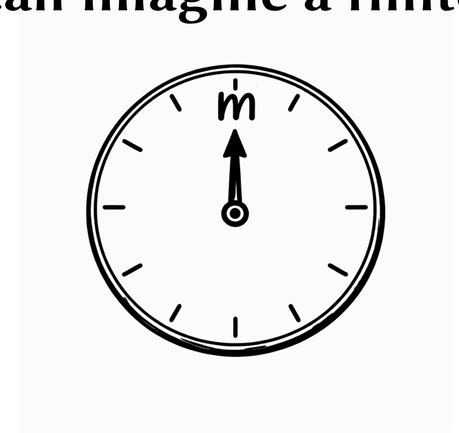
Closure: holds because the range of $\bmod m$ is $[0, m-1] = \mathbb{Z}_m$.

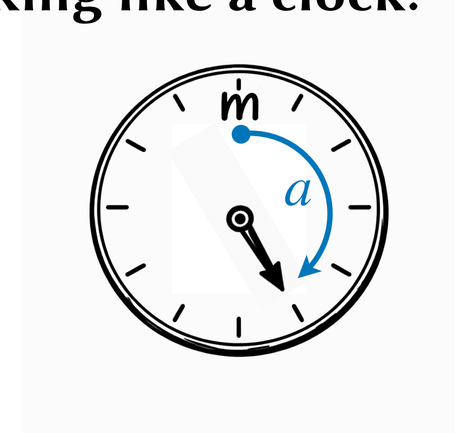
Associativity, Identity, Commutativity: the same as integer $+$ on $\mathbb{Z}$. Identity element is 0.

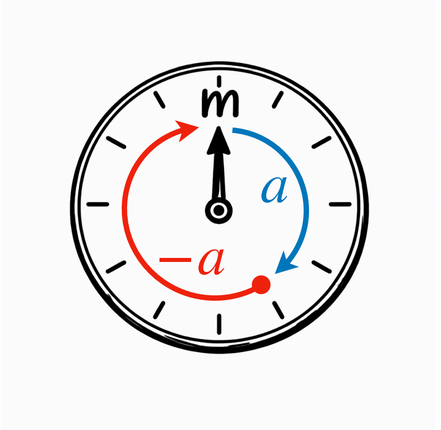
Inverses: Because $0 + 0 = 0$, the additive inverse of 0 is itself.
Notice that $m \bmod m = 0$. The additive inverse of $a \in \mathbb{Z}_m$ is a number $b \in \mathbb{Z}_m$ such that $(a + b) \bmod m = m \bmod m = 0$. Notice that $b \in \mathbb{Z}_m$ always exists.

We will refer to the additive inverse of a as "$-a$". Note that maybe $|a| \neq |-a|$!

You can imagine a finite group working like a clock:







Subgroups!

Definition 10 (subgroups): Let $(\mathbb{G}, \star)$ be a group and let $\mathbb{H} \subseteq \mathbb{G}$ be a subset of $\mathbb{G}$. We say that $(\mathbb{H}, \star)$ is a *subgroup* of $\mathbb{G}$, denoted $\mathbb{H} \leq \mathbb{G}$, if $\mathbb{H}$ is a group wrt the same $\star$ operation as $(\mathbb{G}, \star)$.

Examples: $(\mathbb{Z}, +)$ is a subgroup of $(\mathbb{R}, +)$.

$(2\mathbb{Z}, +)$ where $2\mathbb{Z} = \{2x : x \in \mathbb{Z}\}$ is a subgroup of $(\mathbb{Z}, +)$.

If $+_m$ denotes addition modulo m, then $(\mathbb{Z}_2, +_2)$ is *not* a subgroup of $(\mathbb{Z}_6, +_6)$. *Why not?*

$(\{0,2,4\}, +_6)$ is a subgroup of $(\mathbb{Z}_6, +_6)$, but $(\{0,1,2,4\}, +_6)$ is not. *Why?*

Is $(\{1,2,4\}, \cdot_7)$ *a subgroup of* $(\mathbb{Z}_7^*, \cdot_7)$, *where* $\mathbb{Z}_7^* = \mathbb{Z}_7 \backslash \{0\}$? **Yes.**

Definition 11 (group exponentiation): Let $(\mathbb{G}, \cdot)$ be a group, let $g \in \mathbb{G}$ be any element, and let $g^{-1} \in \mathbb{G}$ be its inverse. Let $1_{\mathbb{G}}$ be the identity element of $(\mathbb{G}, \cdot)$. We will define $g^0 = 1_{\mathbb{G}}$, and for $x \in \mathbb{N}$ we will define $g^x = g \cdot g^{x-1}$ and $g^{-x} = g^{-1} \cdot g^{-(x-1)}$.

Proposition 4: Let $(\mathbb{G}, \cdot)$ be a group, $g \in \mathbb{G}$, and $x, y \in \mathbb{Z}$. We have $g^x \cdot g^y = g^{x+y}$ and $(g^x)^y = g^{x \cdot y}$.

Definition 12 (generated subgroups, generators): Let $(\mathbb{G}, \cdot)$ be a group and let $g \in \mathbb{G}$ be any element. The subgroup *generated* by g is $\langle g \rangle = \{g^x : x \in \mathbb{Z}\}$. We say g is the *generator* of $\langle g \rangle$.

Subgroups!

Definition 11 (group exponentiation): Let $(\mathbb{G}, \cdot\,)$ be a group, let $g \in \mathbb{G}$ be any element, and let $g^{-1} \in \mathbb{G}$ be its inverse. Let $1_{\mathbb{G}}$ be the identity element of $(\mathbb{G}, \cdot\,)$. We will define $g^0 = 1_{\mathbb{G}}$, and for $x \in \mathbb{N}$ we will define $g^x = g \cdot g^{x-1}$ and $g^{-x} = g^{-1} \cdot g^{-(x-1)}$.

Proposition 4: Let $(\mathbb{G}, \cdot\,)$ be a group, $g \in \mathbb{G}$, and $x, y \in \mathbb{Z}$. We have $g^x \cdot g^y = g^{x+y}$ and $(g^x)^y = g^{x \cdot y}$.

Definition 12 (generated subgroups, generators): Let $(\mathbb{G}, \cdot\,)$ be a group and let $g \in \mathbb{G}$ be any element. The subgroup *generated* by g is $\langle g \rangle = \{g^x : x \in \mathbb{Z}\}$. We say g is the *generator* of $\langle g \rangle$.

Examples: Consider $(\mathbb{Z}, +\,)$. Since we are writing it additively, "exponentiation" is multiplication!

We have $\langle 0 \rangle = \{x \cdot 0 : x \in \mathbb{Z}\} = \{0\}$, $\langle 2 \rangle = \{x \cdot 2 : x \in \mathbb{Z}\} = 2\mathbb{Z}$ for example.

Definition 13 (orders of group elements): Let $(\mathbb{G}, \cdot\,)$ be a group and let $g \in \mathbb{G}$ be any element. The *order* of g, denoted $\text{ord}(g)$, is the order of the subgroup generated by g; I.e., $\text{ord}(g) = |\langle g \rangle|$. $\text{ord}(g)$ is also the smallest $x \in \mathbb{N}$ such that $x > 0 \wedge g^x = 1_{\mathbb{G}}$. If no such x exists, then $\text{ord}(g) = \infty$.

Examples: Consider $(\mathbb{Z}_7^*, \cdot_7\,)$. We have $\mathbb{Z}_7^* = \mathbb{Z}_7 \backslash \{0\} = \{1,2,3,4,5,6\}$ and $|\mathbb{Z}_7^*| = 6$.

$\langle 1 \rangle = \{1\}$, so $\text{ord}(1) = |\langle 1 \rangle| = 1$.
$\langle 2 \rangle = \{2,4,1\}$, so $\text{ord}(2) = |\langle 2 \rangle| = 3$. Notice that $2^3 \bmod 7 = 8 \bmod 7 = 1$.
$\langle 3 \rangle = \{3,2,6,4,5,1\} = \mathbb{Z}_7^*$, so $\text{ord}(3) = |\langle 3 \rangle| = 6$. Notice that $3^6 \bmod 7 = 729 \bmod 7 = 1$.

Subgroups!

Definition 13 (orders of group elements): Let $(\mathbb{G}, \cdot)$ be a group and let $g \in \mathbb{G}$ be any element. The *order* of g, denoted $\text{ord}(g)$, is the order of the subgroup generated by g; I.e., $\text{ord}(g) = |\langle g \rangle|$. $\text{ord}(g)$ is also the smallest $x \in \mathbb{N}$ such that $x > 0 \wedge g^x = 1_{\mathbb{G}}$. If no such x exists, then $\text{ord}(g) = \infty$.

Examples: Consider $(\mathbb{Z}_7^*, \cdot_7)$. We have $\mathbb{Z}_7^* = \mathbb{Z}_7 \backslash \{0\} = \{1,2,3,4,5,6\}$ and $|\mathbb{Z}_7^*| = 6$.

$\langle 1 \rangle = \{1\}$, so $\text{ord}(1) = |\langle 1 \rangle| = 1$.
$\langle 2 \rangle = \{2,4,1\}$, so $\text{ord}(2) = |\langle 2 \rangle| = 3$. Notice that $2^3 \bmod 7 = 8 \bmod 7 = 1$.
$\langle 3 \rangle = \{3,2,6,4,5,1\} = \mathbb{Z}_7^*$, so $\text{ord}(3) = |\langle 3 \rangle| = 6$. Notice that $3^6 \bmod 7 = 729 \bmod 7 = 1$.

Definition 14 (cyclic groups): A group $(\mathbb{G}, \cdot)$ is *cyclic* if it is generated by a single element. That is, if there exists some element $g \in \mathbb{G}$ such that $\langle g \rangle = \mathbb{G}$. Note that $\langle g \rangle$ is cyclic by definition.

Example: Consider $(\mathbb{Z}_7^*, \cdot_7)$ again. Since $\langle 3 \rangle = \{3,2,6,4,5,1\} = \mathbb{Z}_7^*$, $(\mathbb{Z}_7^*, \cdot_7)$ must be cyclic.

Lemma 1 (Lagrange's Theorem): Let $(\mathbb{G}, \cdot)$ be a finite group. $\forall g \in \mathbb{G}$, $\text{ord}(g)$ divides $|\mathbb{G}|$.

Proof: Out of scope for this class, but we'll do it in Grad Crypto.

Corollary 1: Let $(\mathbb{G}, \cdot)$ be a group such that $|\mathbb{G}| = m$. For every $g \in \mathbb{G}$, $g^m = g^0 = 1_{\mathbb{G}}$.

Corollary 2: If $(\mathbb{G}, \cdot)$ is a prime-order group, then it is cyclic, and every $g \in \mathbb{G} \backslash \{1_{\mathbb{G}}\}$ generates $\mathbb{G}$.

Homomorphisms and Isomorphisms

Definition 15 (group homomorphism): Let $(\mathbb{G}, \star)$ and $(\mathbb{H}, \diamond)$ be groups. A *homomorphism* from $\mathbb{G}$ to $\mathbb{H}$ is a function $f : \mathbb{G} \to \mathbb{H}$ such that for every $g_1, g_2 \in \mathbb{G}$, we have $f(g_1 \star g_2) = f(g_1) \diamond f(g_2)$. In other words, a homomorphism is a map from $\mathbb{G}$ to $\mathbb{H}$ that preserves the structure of $\mathbb{G}$.

Definition 16 (group isomorphism): Let $(\mathbb{G}, \star)$ and $(\mathbb{H}, \diamond)$ be groups. An *isomorphism* from $\mathbb{G}$ to $\mathbb{H}$ is a homomorphism that is both injective (one-to-one) and surjective (onto). If there exists an isomorphism between $\mathbb{G}$ and $\mathbb{H}$, then we call them *isomorphic,* and denote this $\mathbb{G} \cong \mathbb{H}$.

Intuitively, groups that are isomorphic have *the same structure*, even if their elements and operations are different, and structure-preserving maps exist in both directions.

Example: $(\mathbb{Z}_3^*, \cdot_3)$ and $(\mathbb{Z}_2, +_2)$ are isomorphic. Note that $\mathbb{Z}_3^* = \{1,2\}$ and $\mathbb{Z}_2 = \{0,1\}$.
Consider $f : \mathbb{Z}_3^* \to \mathbb{Z}_2$ such that $f(1) = 0$ and $f(2) = 1$. Clearly f is one-to-one and onto.
Let's verify by exhaustive inspection that it's a homomorphism:

$$f(1 \cdot_3 1) = f(1) = 0 = 0 +_2 0 = f(1) +_2 f(1)$$

$$f(1 \cdot_3 2) = f(2) = 1 = 0 +_2 1 = f(1) +_2 f(2)$$

$$f(2 \cdot_3 2) = f(1) = 0 = 1 +_2 1 = f(2) +_2 f(2)$$

Don’t Panic

- Once again that was way too much to absorb all in one go!
- Don’t worry if it’s uncomfortable. If we need to use this stuff for a proof, I’ll remind you about it.
- The main things you should focus on are understanding intuitively what it means for a group to be *cyclic*, and what *homomorphisms* and *isomorphisms* are, because we will use these ideas to write assumptions and construct basic primitives such as encryption.
- We’re only a few slides away from constructing a OWF. Come back next time!

CS4501 ~~Cryptographic Protocols~~
Intro to Cryptography Speedrun
Lecture 12: Bounded Adversaries OWFs, Assumptions on Groups

https://jackdoerner.net/teaching/#2026/Spring/CS4501