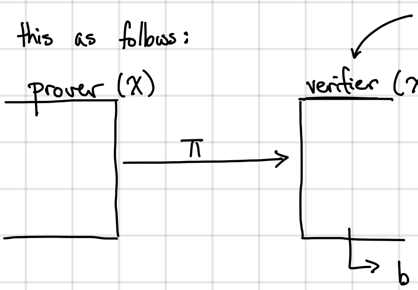
Focus thus far in the course: protecting communication (e.g., message confidentiality and message integrity)

Remainder of course: protecting <u>computations</u>

<u>Zero-knowledge</u>: a defining idea at the heart of theoretical cryptography
↳ Idea will seem very counter-intuitive, but surprisingly powerful ← with surprising implications (DSA/ECDSA signatures based on ZK!)
↳ Showcases the importance and power of definitions (e.g., "What does it mean to know something?")

We begin by introducing the notion of a "proof system"
- <u>Goal</u>: A prover wants to convince a verifier that some <u>statement</u> is true
  e.g., "This Sudoku puzzle has a unique solution"
  "The number N is a product of two prime numbers p and q"
  "I know the discrete log of h base g"
  } these are all examples of statements

We model this as follows:



x: statement that the prover is trying to prove (known to both prover and verifier)
↳ We will write $\mathcal{L}$ to denote the set of <u>true</u> statements (called a "language")

π: the proof of x

↳ $b \in \{0,1\}$ – given statement x and proof π, verifier decides whether to <u>accept</u> or <u>reject</u>

Properties we care about:
- <u>Completeness</u>: Honest prover should be able to convince honest verifier of true statements
$$\forall x \in \mathcal{L} : \Pr[\pi \leftarrow P(x) : V(x,\pi) = 1] = 1$$
[Could relax requirement to allow for some error]
- <u>Soundness</u>: Dishonest prover cannot convince honest verifier of false statement
$$\forall x \notin \mathcal{L} : \Pr[\pi \leftarrow P(x) : V(x,\pi) = 1] < 1/3$$
<u>Important</u>: We are <u>not</u> restricting to <u>efficient</u> provers (for now)

Typically, proofs are "one-shot" (i.e., single message from prover to verifier) and the verifier's decision algorithm is <u>deterministic</u>
↳ Languages with these types of proof systems precisely coincide with NP (proof of statement x is to send NP witness w)

Recall that NP is the class of languages where there is a deterministic solution-checker:

$$\mathcal{L} \in \mathsf{NP} \iff \exists \text{ efficiently-computable relation } R \text{ s.t.}$$
$$\underset{\text{statement}}{x} \in \underset{\text{language}}{\mathcal{L}} \iff \exists \underset{\text{witness}}{w} \in \{0,1\}^{|x|} : \underset{\text{NP relation}}{R}(x,w) = 1$$

<u>Proof system for NP</u>:

$$\text{prover } (x) \xrightarrow{\quad w \quad} \text{verifier } (x)$$

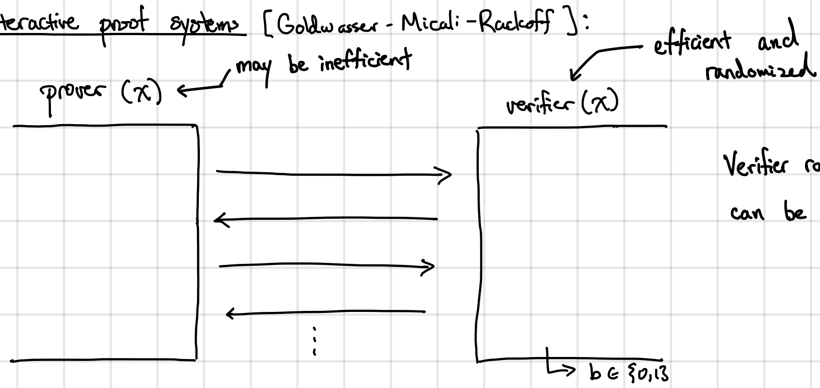
accept if $R(x,w) = 1$

Perfect completeness + soundness

Going beyond NP: we augment the model as follows

- Add randomness: the verifier can be a randomized algorithm
- Add interaction: verifier can ask "questions" to the prover

Interactive proof systems [Goldwasser - Micali - Rackoff]:



Verifier randomness is critical. Otherwise, class of languages that can be recognized collapses to NP. (See HW5).

Interactive proof should satisfy completeness + soundness (as defined earlier)

We define $IP[k]$ to denote class of languages where there is an interactive proof with k messages.
We write $IP = IP[\text{poly}(n)]$ where n is the statement length
(i.e., IP is the class of languages with an interactive proof with polynomially-many rounds)

What is the power of IP?

- For constant number of messages, seems comparable to NP
- Going from constant to polynomial number of rounds is significant!

Theorem. (Lund-Fortnow-Karloff-Nisan '90, Shamir '90) IP = PSPACE. the set of languages that can be checked in polynomial space

Consider following example: Suppose prover wants to convince verifier that $N = pq$ where p, q are prime (and secret).

prover (N, p, q) $\qquad\qquad\qquad\qquad$ verifier (N)

$$\xrightarrow{\pi = (p, q)}$$

$\downarrow$

accept if $N = pq$ and reject otherwise

Proof is certainly complete and sound, but now verifier also learned the factorization of N... (may not be desirable if prover was trying to convince verifier that N is a proper RSA modulus (for a cryptographic scheme) without revealing factorization in the process

↳ In some sense, this proof conveys information to the verifier [i.e., verifier learns something it did not know before seeing the proof]

Zero-knowledge: ensure that verifier does not learn anything (other than the fact that the statement is true)

How do we define "zero-knowledge"? We will introduce a notion of a "simulator."

Definition. An interactive proof system $\langle P, V \rangle$ for a language $\mathcal{L}$ is zero-knowledge if for all efficient (and possibly malicious) verifiers V^*, there exists an efficient simulator S such that for all $x \in \mathcal{L}$:

$$\underbrace{\mathrm{View}_{V^*}(\langle P, V \rangle (x))}_{\substack{\text{random variable denoting the set of messages} \\ \text{sent and received by } V^* \text{ when interacting with the prover } P \text{ on input } x}} \stackrel{c}{\approx} S(x)$$

What does this definition mean?

$\mathrm{View}_{V^*}(P \leftrightarrow V^*(x))$: this is what V^* sees in the interactive proof protocol with P

$S(x)$: this is a function that only depends on the statement x, which V^* already has

If these two distributions are indistinguishable, then anything that V^* could have learned by talking to P, it could have learned just by invoking the simulator itself, and the simulator output only depends on x, which V^* already knows

↳ In other words, anything V^* could have learned (i.e., computed) after interacting with P, it could have learned without ever talking to P!

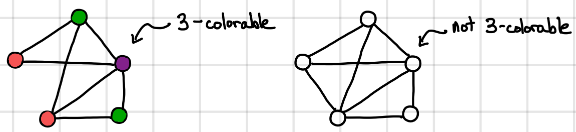
Very remarkable definition!

can in fact be constructed from OWFs

More remarkable: Using cryptographic commitments, then every language $\mathcal{L} \in IP$ has a zero-knowledge proof system.

↳ Namely, anything that can be proved can be proved in zero-knowledge!

We will show this theorem for NP languages. Here it suffices to construct a single zero-knowledge proof system for an NP-complete language. We will consider the language of graph 3-colorability.



3-coloring: given a graph G, can you color the vertices so that no adjacent nodes have the same color?

We will need a commitment scheme. A (non-interactive) commitment scheme consists of three algorithms (Setup, Commit, Open):

cryptographic analog of a sealed "envelope"

- $\text{Setup}(1^\lambda) \to \sigma$: Outputs a common reference string (used to generate/validate commitments) σ
- $\text{Commit}(\sigma, m) \to (c, \pi)$: Takes the CRS σ and message m and outputs a commitment c and opening π
- $\text{Verify}(\sigma, m, c, \pi) \to 0/1$: Checks if c is a valid commitment to m (given π)

Typical setup:

Committer	Verifier
	$\sigma \leftarrow \text{Setup}(1^\lambda)$

$\xleftarrow{\quad \sigma \quad}$

$(c, \pi) \leftarrow \text{Commit}(\sigma, m)$ $\xrightarrow{\quad c \quad}$

(sometime later)

$\xrightarrow{\quad m, \pi \quad}$ can check that $\text{Verify}(\sigma, m, c, \pi) = 1$

Requirements:

- Correctness: for all messages m:

$$\Pr[\sigma \leftarrow \text{Setup}(1^\lambda);\ (c, \pi) \leftarrow \text{Commit}(\sigma, m);\ \text{Verify}(\sigma, c, m, \pi) = 1] = 1$$

- Hiding: for all common reference strings $\sigma \in \{0,1\}^n$ and all efficient A, following distributions are computationally indistinguishable:

adversary	challenger
	$b \in \{0,1\}$

$\xleftarrow{\quad \sigma \quad}$

$\xrightarrow{\quad m_0, m_1 \quad}$ $(c, \pi) \leftarrow \text{Commit}(\sigma, m_b)$

$\xleftarrow{\quad c \quad}$

$b' \in \{0,1\}$

$$\left| \Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1] \right| = \text{negl}(\lambda)$$

- Binding: for all adversaries A, if $\sigma \leftarrow \text{Setup}(1^\lambda)$, then

$$\Pr[(m_0, m_1, c, \pi_0, \pi_1) \leftarrow A : m_0 \neq m_1 \text{ and } \text{Verify}(\sigma, c, m_0, \pi_0) = 1 = \text{Verify}(\sigma, c, m_1, \pi_1)] = \text{negl}(\lambda)$$