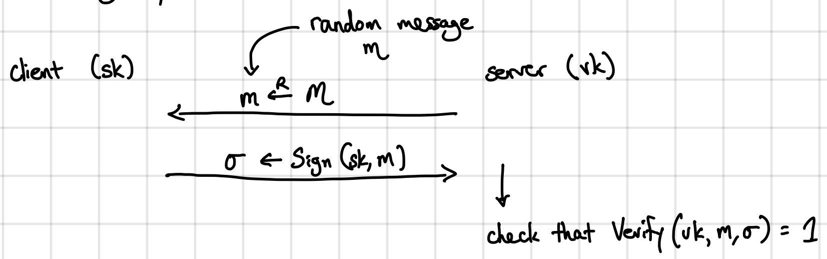
Signature-based challenge-response

- Server stores a verification key vk for digital signature scheme
- Client holds signing key sk


random message
m
client (sk)
server (vk)
$m \xleftarrow{R} \mathcal{M}$
$\sigma \leftarrow \text{Sign}(sk, m)$
check that $\text{Verify}(vk, m, \sigma) = 1$

Server asks client to sign a random message

↳ Client's signature indicates proof of possession of sk associated with vk

↳ Active adversary that interacts with the client before interacting with the prover cannot forge signatures

Provides active security but signatures are long (~ 384 bits)

Signature-based challenge response: client "demonstrates knowledge" of signing key

↳ we will generalize this to "proving" arbitrary statements

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

Remainder of course: protecting computations

Zero-knowledge: 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?")

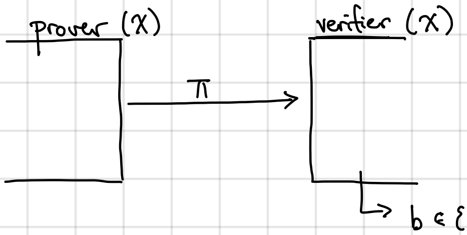
We begin by introducing the notion of a "proof system"

- Goal: A prover wants to convince a verifier that some statement 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:

the verifier is assumed to be an efficient algorithm



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 true statements (called a language)

π : the proof of x

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

Properties we care about:

- Completeness: 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]

- Soundness: Dishonest prover cannot convince honest verifier of false statement

$$\forall x \notin \mathcal{L} : \Pr[\pi \leftarrow P(x) : V(x,\pi) = 1] = \underset{\uparrow \text{ negligible in the statement length}}{\text{negl}(|x|)}$$

Typically, proofs are "one-shot" (i.e., single message from prover to verifier) and the verifier's decision algorithm is deterministic

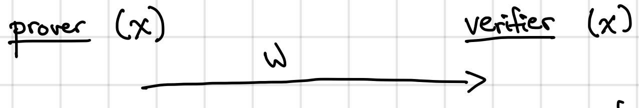
↳ Languages with these types of proof systems precisely coincide with NP (proof of statement x is to send NP witness w)

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

$$\mathcal{L} \in \text{NP} \iff \exists \text{ efficiently-computable relation } R \text{ s.t.}$$

$$\underset{\substack{\nearrow \\ \text{statement}}}{x} \in \underset{\substack{\uparrow \\ \text{language}}}{\mathcal{L}} \iff \exists \underset{\substack{\uparrow \\ \text{witness}}}{w} \in \{0,1\}^{|x|} : \underset{\substack{\uparrow \\ \text{NP relation}}}{R}(x,w) = 1$$

Proof system for NP:



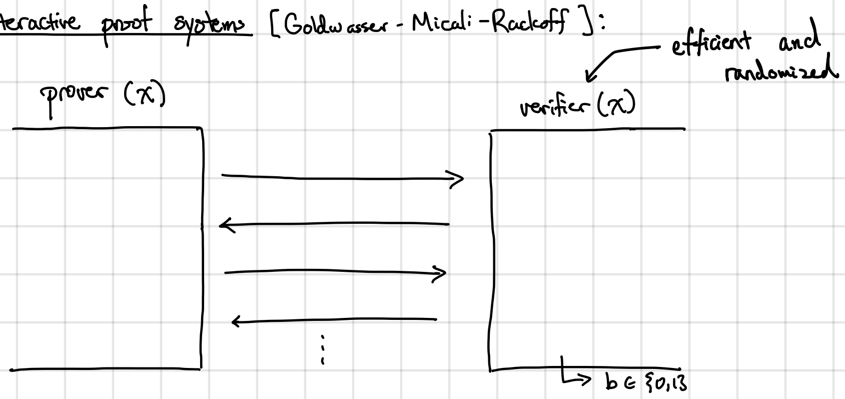
accept if $R(x,w) = 1$

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

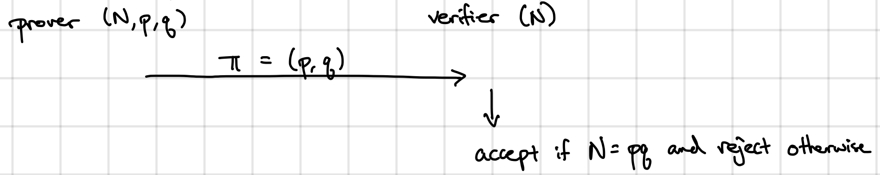
] allows proving statements that are beyond NP

Interactive proof systems [Goldwasser - Micali - Rackoff]:



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

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



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 $x \in \mathcal{L}$:

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

What does this definition mean?

$\text{View}_{V^*}(P \leftrightarrow V^*(x))$: 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!

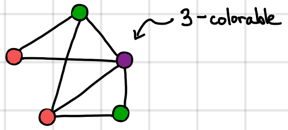
Very remarkable definition!

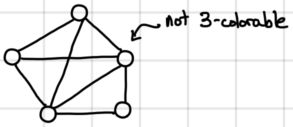
can in fact be constructed from OWFs

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.





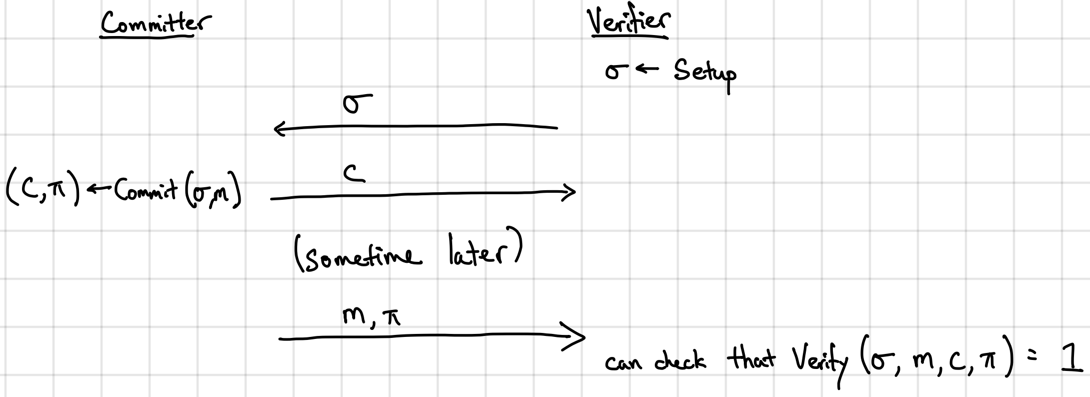
3-coloring: given a graph G, can you color the vertices so that no adjacent nodes have the same color?

cryptographic analog of a sealed "envelope" (see HW4)

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

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

Typical setup:


Committer
Verifier
$\sigma \leftarrow$ Setup
σ
$(c, \pi) \leftarrow$ Commit(σ, m)
c
(sometime later)
m, π
can check that Verify$(\sigma, m, c, \pi) = 1$