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

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

- 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 $x$ base $g$

There are all examples of statements

We model this as follows:

\[
\begin{array}{c}
\text{prover} (X) \\
\text{verifier} (X)
\end{array} \xrightarrow{\text{π}} \begin{array}{c}
\text{π} : \text{the proof of } X \\
\{0,1\}^* : \text{given statement } X \text{ and proof } \pi, \text{ verifier decides whether to accept or reject}
\end{array}
\]

Properties we care about:

- Completeness: Honest prover should be able to convince honest verifier of true statements
  \[\forall X \in L : \Pr [\pi = P(X) : V(X, \pi) = 1] = 1\]

- Soundness: Dishonest prover cannot convince honest verifier of false statement
  \[\forall X \notin L : \Pr [\pi = P(X) : V(X, \pi) = 1] < 1/3\]

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:

\[L \in \text{NP} \iff \exists \text{ efficiently-computable relation } R, \text{ s.t.} \]

\[V X \in L \iff \exists w \in \{0,1\}^* : R(X, w) = 1\]

Proof system for NP:

\[
\begin{array}{c}
\text{prover} (X) \\
\text{verifier} (X)
\end{array} \xrightarrow{\text{R}} \begin{array}{c}
\text{accept if } R(X, w) = 1
\end{array}
\]

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]:

Efficient and randomised

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

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 $IP = IP[po(n)]$ 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) $IP = \text{PSPACE}$.
Consider the following example: Suppose a prover wants to convince a verifier that $N = pq$, where $p, q$ are prime (and secret).

\[
\text{prover} \left( N, (p, q) \right) \quad \vdash \quad \text{accept if } N = pq \text{ and reject otherwise}
\]

To convince the verifier that $N$ is a product of two primes, the prover needs to provide some information that the verifier can check. One way to do this is to randomly select a commitment $r$ and send it to the verifier. The verifier can then compute $r^2 \equiv N$ (mod $pq$) and compare it to the commitment $r$. If they match, the verifier is convinced that $N = pq$.

This is an example of a zero-knowledge proof. Zero-knowledge proofs allow a prover to convince a verifier that a statement is true without revealing any information about the statement itself. This is useful in scenarios where the prover wants to prove something without disclosing private information.

How do we define a zero-knowledge proof? We will introduce a notion of a "simulator" for a language $L$.

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

\[
\Pr[\text{View}_x(\langle P, V \rangle(x))] \approx \Pr[\text{View}_x(\langle P, V^* \rangle(x))]
\]

where $\text{View}_x$ is the set of messages sent and received by $V^*$ when interacting with the prover $P$ on input $x$. What does this definition mean?

- $\text{View}_x(\langle P, V \rangle(x))$: 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 knows.

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!

More remarkable: Using cryptographic commitments, then every language $L \in \text{IP}$ 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):

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

**Typical setup**:

\[
\begin{array}{cccccc}
\text{Committer} & \sigma \leftarrow \text{Setup}(1^n) \\
\downarrow & \downarrow \\
\sigma & c \\
\downarrow & \downarrow \\
(c, \pi) \leftarrow \text{Commit}(\sigma, m) & \text{(Sometime later)} & m, \pi & \text{can check that Verify}(\sigma, m, c, \pi) = 1
\end{array}
\]

**Requirements**:

- **Correctness**: for all messages \(m\):
  \[\Pr[\sigma \leftarrow \text{Setup}(1^n); (c, \pi) \leftarrow \text{Commit}(\sigma, m); \text{Verify}(\sigma, c, m, \pi) = 1] = 1\]

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

\[
\begin{array}{cccccc}
\text{adversary} & \sigma \leftarrow \text{Setup}(1^n) \\
\downarrow & \downarrow \\
\sigma & m_0, m_1 \\
\downarrow & \downarrow \\
\sigma & c \\
\downarrow & \downarrow \\
(c, \pi) \leftarrow \text{Commit}(\sigma, m_b) & \text{challenger} \\
\end{array}
\]

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

- **Binding**: for all adversaries \(A\), if \(\sigma \leftarrow \text{Setup}(1^n)\), then
  \[\Pr[(m_0, m_1, c, \pi) \leftarrow A : m_0 \neq m_1 \text{ and } \text{Verify}(\sigma, c, m, \pi) = 1 = \text{Verify}(\sigma, c, m, \pi)] = \text{negl}(\lambda)\]