Logistics: Problem set 1 due Friday at 5 pm (submit via Gradescope).
Problem set 2 will be posted on Friday.
No office hours on Thursday/Friday — we will try and monitor Piazza.

Previous lecture introduced notion of interactive proofs and zero-knowledge:
- Interactive proof: protocol where prover convinces a verifier that some statement \( \mathcal{X} \) is true.
- Zero-knowledge proof: proof reveals nothing more other than the fact that the statement is true
  (formalized by defining notion of a simulator: anything a verifier could have learned from the protocol execution, it could have also computed itself by running the simulator)

Recap: Zero-knowledge protocol for graph 3-coloring:

Let \( V \) denote the set of vertices in \( G \) and let \( E \) denote the set of edges in \( G \).
Let \( \Phi : V \rightarrow \{0,1,2\} \) be the coloring function.

Completeness: Follow by inspection.

Soundness: If \( G \) is not 3-colorable, then prover did not commit to a valid 3-coloring. At least 1 edge must be "bad," so verifier catches cheating prover with probability \( \frac{1}{|E|} \) on each iteration. Prover succeeds on \( |E| \) iterations with probability at most \( (1 - \frac{1}{|E|})^{|E|} \leq \frac{1}{e} \).
Zero-Knowledge: Construct simulator for verifier \( V^* \) as follows:

1. Simulator commits to random color for every vertex in \( G \).
2. Simulator makes \( V^* \) with committed values. If \( V^* \) queries for an edge \( (u,v) \) where the simulator committed to distinct colors, then the simulator opens those commitments. Otherwise, the simulator “rounds” \( V^* \) to the beginning of the round.
3. The simulator outputs the simulated transcripts for each round of the protocol.

This lecture: Proofs of knowledge

\( \Sigma \)-protocols (Sigma protocols)

In many cases, we want a stronger property: the prover actually "knows" a witness.

\[ \iff \]

Specifically, the prover should know why a statement is true.

For instance, consider the following language:

\[ \mathcal{L} = \{ h = g^x \in G \text{ for some } x \in \mathbb{Z}_q \} = G \]

generator of \( G \)  

In this case, all statements in \( G \) are true (i.e., contained in \( \mathcal{L} \)), but we can still consider a notion of proving knowledge of the discrete log of an element \( h \in G \) — conceptually stronger property than proof of membership.

**Question:** What does it mean to "know" something? [GMR85]

If a prover is able to convince an honest verifier that it "knows" something, then it should be possible to extract that quantity from the prover.

**Def:** An interactive proof system \( (P,V) \) is a proof of knowledge for an NP relation \( R \) if there exists an efficient extractor \( E \) such that for any \( x \) and any prover \( P^* \):

\[
Pr[w \leftarrow E^*(x) : R(x,w) = 1] \geq Pr[(P^*,V)(x) = 1] - \varepsilon
\]

more generally,  

\[ \varepsilon \] knowledge error

Trivial proof of knowledge: prover sends witness in the clear to the verifier.

\[ \iff \]

In most applications, we additionally require zero-knowledge.

**Note:** knowledge is a strictly stronger property than soundness.

\[ \iff \]

if protocol has knowledge error \( \varepsilon \) it also has soundness error \( \varepsilon \) (i.e., a dishonest prover convinces an honest verifier of a false statement with probability at most \( \varepsilon \)).
Proving knowledge of discrete log ( Schnorr’s protocol)

Suppose prover wants to prove it knows x such that \( h = g^x \) (i.e. prover demonstrates knowledge of discrete log of h base g)

\[ \begin{align*}
\text{prover} & \quad r \leftarrow Z_q \\
& \quad u \leftarrow g^r \\
& \quad z \leftarrow r + cx
\end{align*} \]

\[ \begin{align*}
& \quad c \leftarrow Z_q \\
& \quad e \leftarrow Z_q
\end{align*} \]

\[ \begin{align*}
& \quad z \leftarrow r + cx \\
& \quad v \leftarrow u^e \cdot Z_q^c
\end{align*} \]

Completeness: if \( z = r + cx \), then

\[ g^z = g^{r + cx} = g^r g^{cx} = u \cdot h^c \]

Honest-Verifier Zero-Knowledge: build a simulator as follows (familiar strategy: run the protocol in “reverse”):

on input \((g, h)\):

1. Sample \( z \leftarrow Z_q \) \quad \text{(uniformly random challenge)}
2. Sample \( c \leftarrow Z_q \) \quad \text{(chosen so that }
3. Set \( u = g^{z/c} \) and output \((u, c, z)\)

Simulated transcript is identically distributed as the real transcript with an honest verifier.

Question: What goes wrong if the challenge \( c \) is not chosen uniformly at random?

Above simulation no longer works (since we cannot sample \( z \) first).

\[ \Rightarrow \quad \text{To get general zero-knowledge, we require that the verifier first commit to its challenge (using a statistically hiding commitment).} \]

Knowledge: Suppose \( P^* \) is (possibly malicious) prover that convinces honest verifier with probability 1. We construct an extractor as follows:

1. Run the prover \( P^* \) to obtain an initial message \( u \).
2. Send a challenge \( c \leftarrow Z_q \) to \( P^* \). The prover replies with a response \( z_i \).
3. “Rewind” the prover \( P^* \) so its internal state is the same as it was at the end of Step 1. Then, send another challenge \( c^2 \leftarrow Z_q \) to \( P^* \). Let \( z_2 \) be the response of \( P^* \).
4. Compute and output \( \chi = (z_2 - z_1)(c_2 - c_1)^{-1} \in Z_q \).

Since \( P^* \) succeeds with probability 1 and the extractor perfectly simulates the honest verifier’s behavior, with probability 1, both \((u, c_1, z_1)\) and \((u, c_2, z_2)\) are both accepting transcripts. This means that

\[ g^{z_1} = u \cdot h^{c_1} \quad \text{and} \quad g^{z_2} = u \cdot h^{c_2} \]

\[ \Rightarrow \quad \frac{g^{z_1}}{h^{c_1}} = \frac{g^{z_2}}{h^{c_2}} \quad \Rightarrow \quad g^{z_1 + c_1 \chi} = g^{z_2 + c_2 \chi} \]

Thus, extractor succeeds with overwhelming probability.
If $P^*$ succeeds with probability $\varepsilon$, then need to rely on "Rewinding Lemma" to argue that extractor obtains two accepting transcripts with probability at least $\varepsilon^2 - \frac{1}{8}$.

Intuitively: extractor succeeds by forcing prover to answer multiple challenges (via rewinding). However, the real verifier cannot rewind the prover, so the real verifier cannot extract the secret by interacting with the prover.

Identification Protocol for Discrete Log

Suppose a client wants to authenticate to the server

1) Goal: security against active adversaries (adversary sees contents of the server and can interact arbitrarily with the client)

Can directly build such a scheme from Schnorr's protocol:

\[
\begin{array}{c}
\text{client (X)} \quad \text{secret (endowed)} \quad \text{server (g, h = g^2)} \\
\hline
\end{array}
\]

protocol is precisely 3-round

Schnorr proof of knowledge of discrete log

Correctness of this protocol follows from completeness of Schnorr's protocol

(Active) security follows from knowledge property and zero-knowledge

Intuitively: knowledge says that any client that successfully authenticates must know secret $X$

Zero-knowledge says that interactions with honest client (i.e., the prover) do not reveal anything about $X$

(for active security, require protocol that provides general zero-knowledge rather than just AVZK)

More general view: $\Sigma$-protocols (Sigma protocols)

Prover (X) \quad g, h = g^X \quad \rightarrow \text{"commitment"} \quad \text{Verifier}

\[
\begin{array}{c}
\downarrow c \\
\leftarrow c' \quad \text{"challenge"} \quad \text{(random string, "public-coins")} \\
\downarrow r + cX \\
\text{"response"} \quad \text{protocol flow resembles a $\Sigma$}
\end{array}
\]

Protocols with this structure (commitment-challenge-response) are called $\Sigma$-protocols (Sigma protocols)

Many variants of Schnorr protocols can be used to prove knowledge of statements like:

- Common discrete log: $X$ such that $h_1 = g^x$ and $h_2 = g^y$ (useful for building a verifiable random function)
- DDH tuple: $(g, u, v, w)$ is a DDH tuple - namely, that $u = g^a$, $v = g^b$, and $w = g^{ab}$ for $a, b \in \mathbb{Z}_q$

L\text{\textguillemotleft} useful for proving relations on ElGamal ciphertexts (e.g., that a particular ElGamal ciphertext encrypts either 0 or 1)

L\text{\textguillemotleft} useful building block in constructions of DDH-based oblivious transfer (OT) protocols - Naor-Pinkas (more details next lecture)

L\text{\textguillemotleft} Reduces to proving common discrete log: $(g, u, v, w)$ is a DDH tuple if and only if there is an $X$ such that $v = g^x$ and $w = u^x$
Showing that $h_i = g_i^X$ and $h_2 = g_2^X$:

\[
\begin{align*}
\text{prover} & \quad r \in \mathbb{Z}_q^* \quad u_i = g_i^r \quad u_i = g_i^r \quad t \in \mathbb{Z}_q^* \quad t = r + x \quad Z = r + t \times X \\
\text{verifier} & \quad \text{check that } g_i^Z = u_i \cdot h_i^t \quad \text{and} \quad g_2^Z = u_2 \cdot h_2^t
\end{align*}
\]

Completeness and HVZK follows as in Schnorr's protocol.

**Knowledge**: Two scenarios:

1. If prover does inconsistent commitment (i.e., $u_i = g_i^r$ and $u_i = g_i^r$ where $r_i \neq r_2$), then over choice of honest verifier's randomness, then prover can only succeed with probability at most $1/2$:

   \[
   Z = r_1 + x \cdot t = r_2 + x \cdot t \quad (\text{if verifier accepts})
   \]

   This means that
   \[
   (r_1 - r_2) = t (x_1 - x_2)
   \]

   If $r_i \neq r_2$, there is at most $1/t \in \mathbb{Z}_q^*$ where this relation holds. Since $t$ is uniform over $\mathbb{Z}_q$, the verifier accepts with probability at most $1/2$.

2. If prover succeeds with $1/2$ probability, then it must use a "consistent" commitment. Can build extractor just as in Schnorr's protocol. Knowledge error larger by additive $1/2$ term (from above analysis).

Next lecture: $\Sigma$-protocols $\Rightarrow$ non-interactive zero-knowledge protocols and signatures.