Non-interactive zero-knowledge: Can we construct a zero-knowledge proof system where the proof is a single message from the prover to the verifier? 

\[
\begin{array}{c}
\text{prover} (x, \omega) \\
\uparrow \\
\Pi = \text{Prove}(x, \omega) \\
\downarrow \\
\text{verifier} (x) \\
\end{array}
\]

Why do we care? Interaction in practice is expensive!

Unfortunately, NIZKs are only possible for sufficiently easy languages (i.e., languages in BPP).

- The simulator (for 2K property) can essentially be used to decide the language:
  - if \( x \in L \): \( S(x) \rightarrow \Pi \) and \( x \) should be accepted by the verifier (by 2K)
  - if \( x \notin L \): \( S(x) \rightarrow \Pi \) but \( x \) should not be accepted by verifier (by soundness)

\( \text{NIZK impossible for NP unless } \NP \subseteq \BPP \) (whide!)

Impossibility results tell us where to look! If we cannot succeed in the "plain" model, then move to a different one:

- common random/reference string (CRS) model:

  ![Diagram](http://example.com/diagram.png)

  - prover and verifier have access to shared randomness
  - could be a uniformly random string or a structured string

- random oracle model:

  ![Diagram](http://example.com/diagram.png)

  - in this model, simulator can "program" the random oracle (again, asymmetry between real prover and the simulator)

\( \Rightarrow \) In both cases, simulator has additional "power" than the real prover, which is critical for enabling NIZK constructions for NP.

Esc-Schnorr heuristics: from Z-protocols to NIZK in RO model

Recall Schnorr’s protocol for proving knowledge of discrete log:

\[
\begin{array}{ccc}
\text{prover} & \xleftarrow{\sim} & \text{verifier} \\
\omega & \leftarrow & \gamma^x \\
\text{verify that } & \gamma^x = u \cdot h^c
\end{array}
\]

In this protocol, verifier’s message is uniformly random (and in fact, is "public coin" — the verifier has no secrets)

Key idea: Replace the verifier’s challenge with a hash function \( H: [0,1]^n \rightarrow \mathbb{Z}_p \)

- Namely, instead of sampling \( c \in \mathbb{Z}_p \), we sample \( c = H(y, h, u) \). \( \leftarrow \) prover can now compute this quantity on its own!

Completeness, zero-knowledge, proof of knowledge follow by a similar analysis as Schnorr [will rely on random oracle]
Signatures from discrete log in RO model (Schnorr):

- Setup: \( x \in \mathbb{Z}_p \)
  \[ \text{vk}: (g, h = g^x) \quad \text{sk}: x \]

- Sign \((sk, m)\): \( r \in \mathbb{Z}_p \)
  \[ u \leftarrow g^r, \quad c \leftarrow H(g, h, u, m), \quad z \leftarrow r + cx \]
  \[ \sigma = (u, z) \]

- Verify \((vk, m, \sigma)\): write \( \sigma = (u, z) \), compute \( c \leftarrow H(g, h, u, m) \) and accept if \( g^z = u \cdot h \)

Security essentially follows from security of Schnorr's identification protocol (together with Fiat-Shamir)

\[ \rightarrow \text{forged signature on a new message } m \text{ is a proof of knowledge of the discrete log (can be extracted from adversary)} \]

More generally, any \( \Sigma \)-protocol can be used to build a signature scheme using the Fiat-Shamir heuristic (by using the message to derive the challenge via RO)

Length of Schnorr's signature: \( \text{vk}: (g, h = g^x) \) \( \sigma: (g^z, c = H(g, h, g^z, m), z = r + cx) \)
Verification checks that \( g^z = g^x \)

But, can do better... observe that challenge \( c \) only needs to be 128 bits (the knowledge error of Schnorr is \( \frac{1}{16} \) where \( c \) is the set of possible challenges), so we can sample a 128-bit challenge rather than 256-bit challenge. Thus, instead of sending \((g^z, z)\), instead send \((c, z)\) and compute \( g^z = g^x \) and that \( c = H(g, h, g^z, m) \). Then resulting signatures are \( 128 \text{ bits} \)

Important note: Schnorr signatures are randomized, and security relies on having good randomness

\[ \rightarrow \text{What happens if randomness is reused for two different signatures?} \]

Then, we have

\[ \sigma_1 = (g^z, c_1 = H(g, h, g^z, m), z_1 = r + cx) \]
\[ \sigma_2 = (g^z, c_2 = H(g, h, g^z, m), z_2 = r + cx) \]
\[ z_1 - z_2 = (c_1 - c_2)x \quad \Rightarrow \quad x = (c_1 - c_2)^{-1}(z_1 - z_2) \]

This is precisely the set of relations the knowledge extractor uses to recover the discrete log \( x \) (ie, the signing key)!
Deterministic Schnorr: We want to replace the random value \( r \in \mathbb{Z}_p \) with one that is deterministic, but which does not compromise security.

\[ \Rightarrow \text{Derive randomness from message using a PRF. In particular, signing key includes a secret PRF key } \hbar, \text{ and } \]

\[ \text{signing algorithm computes } r \leftarrow F(k,m) \text{ and } \sigma \leftarrow \text{Sign}(sk,m;r). \]

\[ \Rightarrow \text{Avoids randomness reuse/mismatch vulnerabilities.} \]

In practice, we use a variant of Schnorr’s signature scheme called \( \text{DSA/ECDSA} \) larger signatures (2 group elements - 512 bits) and proof only in “generic group” model \([\text{but we use it because Schnorr was patented... until 2008}]\)

ECDSA signatures (over a group \( G \) of prime order \( p \)):

- Setup: \( \alpha \in \mathbb{Z}_p \)
  \( \text{vk} : (g, h = g^\alpha) \quad \text{sk} : \alpha \) deterministic function

- Sign (sk, m): \( r \leftarrow f(u) \in \mathbb{Z}_p \)
  \( u \leftarrow g^r \)
  \( \sigma = (r, s) \)

- Verify (vk, m, q):
  \( \sigma = (r, s) \), compute \( u \leftarrow g^{H(m)/s} h^{r/\ell} \), accept if \( r = f(u) \)

Correctness: \( u = g^{H(m)/s} h^{r/\ell} = [H(m) + rX]/\ell = g [H(m) + rX]/(H(m) + rX) \alpha^{-1} = g^\alpha \) and \( r = f(g^\alpha) \)

Security analysis: non-trivial: requires either strong assumptions or modeling \( G \) as an “ideal group”

Signature size: \( \sigma = (r, s) \in \mathbb{Z}_p^2 \) - for 128-bit security, \( p \approx 2^{256} \) so \( |\sigma| = 512 \text{ bits} \) (can use P-256 or Curve 25519)