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?

\[
\begin{array}{c}
\text{prover}(x) \\
\xrightarrow\quad \text{verifier}(x) \\
\downarrow \\
\text{s} \in \{0,1\}^3
\end{array}
\]

Why do we care? Interaction in practice is expensive!

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

\[ \exists \text{ The simulator (for 2K property) can essentially be used to decide the language:} \\
\begin{align*}
\text{if } x \in L : S(x) &\Rightarrow \pi \quad \text{and } \pi \text{ should be accepted by the verifier (by 2K)} \\
\text{if } x \notin L : S(x) &\Rightarrow \pi \quad \text{but } \pi \text{ should not be accepted by verifier (by soundness)} \\
\end{align*}
\]

\[ \text{NIZK impossible for } \text{NP unless } \text{NP} \subseteq \text{BPP (unlikely!)} \]

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:**
  
  In this model, simulator is allowed to choose (i.e., simulate) the CRS in conjunction with the proof, but soundness is defined with respect to a honestly-generated CRS (asymmetry between the capabilities of the real prover and the simulator)

- **Random oracle model:**
  
  In this model, simulator can "program" the random oracle [again, asymmetry between real prover and the simulator]

\[ \Rightarrow \text{In both cases, simulator has additional ''power'' than the real prover, which is critical for enabling NIZK constructions for NP.} \]

**In CRS model:** CRS sampled from Setup(\(\lambda \))

Simulator is able to choose CRS

- Must be computationally indistinguishable from real CRS
- Simulated CRS will typically have a simulator trapdoor that can be used to simulate proofs

Real protocol: CRS is sampled by a trusted party (essential for soundness)

Zero-knowledge says that a particular choice of (CRS, \(\pi\)) can be simulated given only the statement \(x\)

**In random oracle model:** Simulator has ability to program random oracle - must properly simulate distribution of random oracle outputs

Can extend to NIZK proofs of knowledge
Fiat-Shamir heuristics: NIZKs in random oracle model

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

\[
\begin{array}{ccc}
\text{prover} & \text{verifier} \\
\begin{pmatrix} g^r & z \\ h^r & \sqrt{} \\ \sqrt{} \end{pmatrix} & \begin{pmatrix} u & \sqrt{} \\ c & \sqrt{} \end{pmatrix} & \begin{pmatrix} c^r & \sqrt{} \\ \sqrt{} \end{pmatrix} \\
\end{array}
\]

\[
\begin{align*}
r \leftarrow Z_p \\
\text{u} \leftarrow g^r \\
\text{c} \leftarrow H(g, h, u) \\
\text{z} \leftarrow r + cx \\
\text{verify that } g^z = u \cdot h^c
\end{align*}
\]

Key idea: Replace the verifier's challenge with a hash function \( H : \{0,1\}^n \rightarrow Z_p \)

Namely, instead of sampling \( c \leftarrow Z_p \), we sample \( c = H(g, h, u) \). \( u \) prover can now compute this quantity on its own!

Completeness, zero-knowledge, proof of knowledge follows by a similar analysis as Schnorr [will rely on random oracle]

Signatures from discrete log in RO model (Schnorr):
- Setup: \( \times \leftarrow Z_p \)
  - \( \text{vk} : (g, h = g^x) \)
  - \( \text{sk} : x \)
- Sign (sk, m):
  - \( c \leftarrow Z_p \)
  - \( u \leftarrow g^r \)
  - \( c \leftarrow H(g, h, u, m) \)
  - \( z \leftarrow r + cx \)
  - \( \sigma = (u, z) \)
- Verify (vk, m, \sigma):
  - Write \( \sigma = (u, z) \), compute \( c = H(g, h, u, m) \) and accept if \( g^z = u \cdot h^c \)

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

\( \Rightarrow \) forged signature on a new message \( m \) is a proof of knowledge of the discrete log (can be extracted from adversary)

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

\( \Rightarrow \) Verification checks that \( g^z = g^r h^c \)

But, can do better... observe that challenge \( c \) only needs to be 128-bit (the knowledge error of Schnorr is \( \frac{1}{16} \) when \( c \) is the set of possible challenges), so we can sample a 128-bit challenge rather than 256-bit challenge. Thus, instead of sending \( (g^r, z) \), instead send \( (c, z) \) and compute \( g^r = g^{z / c} \) and that \( c = H(g, h, g^r, m) \). Then resulting signatures are 384 bits

128-bit challenge \( c \)
256-bit group element

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

\( \Rightarrow \) What happens if randomness is reused for two different signatures?

Then, we have

\[
\begin{align*}
\sigma_1 &= (g^r, c_1 = H(g, h, g^{r_1}, m), z_1 = r + c_1x) \\
\sigma_2 &= (g^r, c_2 = H(g, h, g^{r_2}, m), z_2 = r + c_2x) \\
\end{align*}
\]

\[
\begin{align*}
z_1 - z_2 &= (c_1 - c_2)x \\
\Rightarrow x &= (c_1 - c_2)^{-1}(z_1 - z_2)
\end{align*}
\]

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

\( \rightarrow \) Derive randomness from message using a PRF. In particular, signing key includes a secret PRF key \( h \), and signing algorithm computes \( r \leftarrow F(k, m) \) and \( \sigma = \text{Sign}(sk, m; r) \).

\( \rightarrow \) Avoids randomness reuse/misuse vulnerabilities.

In practice, we use a variant of Schnorr's signature scheme called DSA/ECDSA.

Larger signatures (2 group elements = 512 bits) and proof only in "generic group" model [but we use it because Schnorr was patented ... until 2008] digital signature algorithm / elliptic-curve DSA

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

- **Setup**: \( \forall g \in \mathbb{Z}_p \)
  - \( \text{vk} = (g, h = g^x) \)
  - \( \text{sk} = X \) deterministic function

- **Sign**(sk, m):
  - \( x \leftarrow \mathbb{Z}_p \)
  - \( r \leftarrow f(u) \in \mathbb{Z}_p \)
  - \( s \leftarrow (H(m) + r \cdot x) / x \in \mathbb{Z}_p \)
  - \( \sigma = (r, s) \)

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

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

**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 \( \lvert \sigma \rvert = 512 \text{ bits} \) (can use P-256 or Curve25519)
An application of zero-knowledge proofs to encrypted voting (based on ElGamal)

\[ pk = (g, h, g^x) \]
\[ sk = x \]

Suppose votes are 0/1. Parties encrypt vote \( v \in \{0, 1\} \) as
\[(g^r, h^v \cdot g^r) \text{ where } r \in \mathbb{Z}_p \]

Votes can be aggregated by computing
\[ \prod_{i \in \{0, 1\}} g^{r_i}, \prod_{i \in \{0, 1\}} h^{v_i} \cdot g^{r_i} \]
\[ \Rightarrow \text{ brute force discrete log to obtain } \Sigma v_i \]

But malicious voter can encrypt \(-1000 : (g^r, h^v \cdot g^r^{-1000})\).

Solution: require voters to provide ZK proof that encrypted vote \((u, v)\) is valid:
\( \text{either } (g^v, h^u, u) \text{ is a DDH tuple OR } (g^v, h^u, u, v/g) \text{ is a DDH tuple} \)
\( \Rightarrow \text{proof using Chaum-Pedersen along with OR proof construction (not discussed here)} \)

Basic approach generalizes to arbitrary ranges.

Fancier versions of these types of ZKPs are used in private telemetry system by Mozilla (Privacy).  

Identification protocol from discrete log:

\[ \text{client (x): secret (exponential)} \rightarrow \text{server (g, h = g^x)} \]
\[ \Rightarrow \text{Schnorr proof of knowledge of discrete log} \]
\[ \text{protocol is precisely 3-round} \]
\[ \text{Essentially, the discrete log of } h \text{ (base } g) \text{ is the client's "password" and instead of sending the password in the clear to the server, the client proves in zero-knowledge that it knows } x \]
\[ \text{Can be made non-interactive via Fiat-Shamir} \]

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 server) do not reveal anything about \( x \)
(for active security, require protocol that provides general zero-knowledge rather than just HVZK)