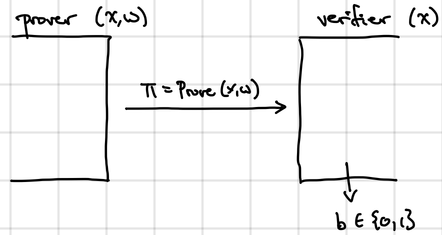
(NIZK)

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?



Why do we care? Interaction in practice is expensive!

Unfortunately, NIZKs are only possible for sufficiently-easy languages (i.e., languages in BPP). — languages that can be decided by a randomized polynomial-time algorithm (w.h.p.)

↳ The simulator (for ZK property) can essentially be used to decide the language

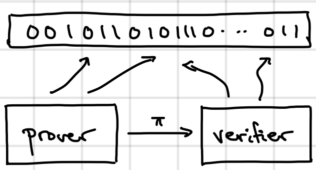
if $x \in \mathcal{L}$: $S(x) \to \pi$ and π should be accepted by the verifier (by ZK)

if $x \notin \mathcal{L}$: $S(x) \to \pi$ but π should not be accepted by verifier (by soundness)

} NIZK impossible for NP unless NP $\subseteq$ BPP (unlikely!)

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

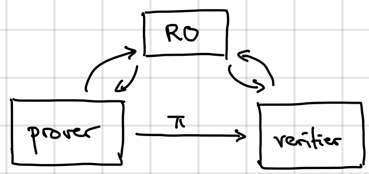
Common random/reference string (CRS) model:



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

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 an 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]

$\Longrightarrow$ 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 $\text{Setup}(1^\lambda)$

Simulator is able to choose CRS

- Must be computationally indistinguishable from real CRS
- Simulated CRS will typically have a simulation 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, π) 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 heuristic: NIZKs in random oracle model

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

prover $(g, h = g^x, x)$ — verifier (g, g^x)

Prover: $r \xleftarrow{R} \mathbb{Z}_p$, $u \leftarrow g^r$ $\quad \xrightarrow{u}$

Verifier: $c \xleftarrow{R} \mathbb{Z}_p$ $\quad \xleftarrow{c}$

Prover: $z \leftarrow r + cx$ $\quad \xrightarrow{z}$

verify that $g^z = u \cdot h^c$

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 $H: \{0,1\}^* \to \mathbb{Z}_p$

Namely, instead of sampling $c \xleftarrow{R} \mathbb{Z}_p$, we sample $c \leftarrow H(g, h, u)$. ← 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: $x \xleftarrow{R} \mathbb{Z}_p$
  $vk: (g, h = g^x) \qquad sk: x$
- Sign (sk, m): $r \xleftarrow{R} \mathbb{Z}_p$
  $u \leftarrow g^r \qquad c \leftarrow H(g, h, u, m) \qquad z \leftarrow r + cx$
  $\sigma = (u, z)$

  } signature is a NIZK proof of knowledge of discrete log of h (with challenge derived from the message m)
- Verify (vk, m, σ): write $\sigma = (u, z)$, $vk = h$, compute $c \leftarrow H(g, h, u, m)$ and accept if $g^z = u \cdot h^c$

Security essentially follows from security of Schnorr's identification protocol (together with Fiat-Shamir)
↳ 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: $vk: (g, h = g^x)$, $sk: x$ $\qquad \sigma: (g^r, \underbrace{c = H(g, h, g^r, m)}_{\text{can be computed given other components, so do not need to include}}, z = r + cx)$ $\qquad$ verification checks that $g^z = g^r h^c$

$\Rightarrow |\sigma| = 2 \cdot |\mathbb{G}|$ [512 bits if $|\mathbb{G}| = 2^{256}$]

But, can do better... observe that challenge c only needs to be 128-bits (the knowledge error of Schnorr is $1/|C|$ 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^r, z), instead send (c, z) and compute $g^r = g^z / h^c$ and that $c = H(g, h, g^r, m)$. Then resulting signatures are 384 bits

128 bit challenge + 256 bit group element

Important note: Schnorr signatures are randomized, and security relies on having good randomness
↳ What happens if randomness is reused for two different signatures?

Then, we have

$$\left.\begin{aligned} \sigma_1 &= (g^r, c_1 = H(g, h, g^r, m_1), z_1 = r + c_1 x) \\ \sigma_2 &= (g^r, c_2 = H(g, h, g^r, m_2), z_2 = r + c_2 x) \end{aligned}\right\} \quad z_1 - z_2 = (c_1 - c_2)x \Rightarrow x = (c_1 - c_2)^{-1}(z_1 - z_2)$$

This is precisely the set of relations the knowledge extractor uses to recover the discrete log x (i.e., the signing key)!

<u>Deterministic Schnorr</u>: We want to replace the random value $r \xleftarrow{R} \mathbb{Z}_p$ with one that is deterministic, but which does not compromise security

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

↳ Avoids randomness reuse/misuse vulnerabilities.

In practice, we (TLS protocol) use a variant of Schnorr's signature scheme called $\overbrace{\text{DSA / ECDSA}}^{\text{digital signature algorithm / elliptic-curve DSA}}$

↳ larger signatures (2 group elements – 512 bits) and proof only in "generic group" model

[but we use it because Schnorr was patented ... until 2008]

ECDSA signatures (over a group $\mathbb{G}$ of prime order p):

- Setup: $x \xleftarrow{R} \mathbb{Z}_p$

  vk: $(g, h = g^x)$ sk: x

- Sign(sk, m): $\alpha \xleftarrow{R} \mathbb{Z}_p$

  $u \leftarrow g^\alpha$ $r \leftarrow f(u) \in \mathbb{Z}_p$ (deterministic function specified by ECDSA)

  $s \leftarrow (H(m) + r \cdot x)/\alpha \in \mathbb{Z}_p$

  $\sigma = (r, s)$

  [specifically, $f(u)$ parses $u = (\hat{x}, \hat{y}) \in \mathbb{F}_q^2$ where $\mathbb{F}_q$ is the base field over which the elliptic curve is defined, and outputs $\hat{x} \pmod p$, where $\hat{x}$ is viewed as a value in $[0, q)$]

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

  $\text{vk} = h$

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

Security analysis non-trivial: requires either strong assumptions or modeling $\mathbb{G}$ as an "ideal" group

Signature size: $\sigma = (r, s) \in \mathbb{Z}_p^2$ — for 128-bit security, $p \sim 2^{256}$ so $|\sigma| = 512$ 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 $t \in \{0,1\}$ as
$$(g^r,\ h^r \cdot g^t) \quad \text{where } r \xleftarrow{R} \mathbb{Z}_p$$

Votes can be aggregated by computing

$$\prod_{i \in [n]} g^{r_i},\ \prod_{i \in [n]} h^{r_i} g^{t_i}$$
$$\quad \| \qquad\qquad \|$$
$$g^{\sum r_i},\ h^{\sum r_i} g^{\sum t_i}$$

decryption recovers $g^{\sum t_i}$

↳ brute force discrete log to obtain $\sum v_i$

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

<u>Solution</u>: require voters to provide ZK proof that encrypted vote (u, v) is valid:

either (g, h, u, v) is a DDH tuple <u>OR</u>
$(g, h, u, v/g)$ is a DDH tuple

} prove using Chaum-Pedersen along with OR proof construction (not discussed here)

↳ can make non-interactive via Fiat-Shamir

Basic approach generalizes to <u>arbitrary</u> ranges.

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

<u>Identification protocol from discrete log</u>:

client (x) — client's secret (credential)

server $(g, h = g^x)$ — public verification key

$$\text{client} \longrightarrow \text{server}$$
$$\text{client} \longleftarrow \text{server}$$
$$\text{client} \longrightarrow \text{server}$$

protocol is precisely 3-round Schnorr proof of knowledge of discrete log → Can be made non-interactive via Fiat-Shamir

Essentially, the discrete log of h (base g) is the client's "password" and instead of sending the password in the <u>clear</u> to the server, the client proves in zero-knowledge that it knows x

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

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

↳ <u>Intuitively</u>: 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 HVZK)