In many cases, we want a stronger property: the prover actually "knows" why a statement is true (e.g., it knows a "witness")

For instance, consider the following language:

$$\mathcal{L} = \{h \in \mathbb{G} \mid \exists x \in \mathbb{Z}_p : h = g^x\} = \mathbb{G}$$

group of order p ; generator of $\mathbb{G}$

Note: this definition of $\mathcal{L}$ implicitly defines an NP relation $\mathcal{R}$:
$\mathcal{R}(h, x) = 1 \iff h = g^x \in \mathbb{G}$

In this case, all statements in $\mathbb{G}$ 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 $h \in \mathbb{G}$ — conceptually stronger property than proof of membership

Philosophical question: What does it mean to "know" something?

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.

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

proof of knowledge is parameterized by a specific relation $\mathcal{R}$ (as opposed to the language $\mathcal{L}$)

$$\Pr[w \leftarrow \mathcal{E}^{P^*}(x) : \mathcal{R}(x, w) = 1] \geq \Pr[\langle P^*, V \rangle (x) = 1] - \varepsilon$$

more generally, could be polynomially smaller

knowledge error

Trivial proof of knowledge: prover sends witness in the clear to the verifier
↳ In most applications, we additionally require zero-knowledge

Note: knowledge is a strictly stronger property than soundness
↳ if protocol has knowledge error $\varepsilon \Rightarrow$ it also has soundness error ε (i.e. a dishonest prover convinces an honest verifier of a false statement with probability at most ε)

Proving knowledge of discrete log (Schnorr's protocol)

assume $g, h \in \mathbb{G}$ where $\mathbb{G}$ has prime order q

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)

prover		verifier
$r \xleftarrow{R} \mathbb{Z}_p$, $u \leftarrow g^r$	$\xrightarrow{u}$	
	$\xleftarrow{c}$	$c \xleftarrow{R} \mathbb{Z}_p$
$z \leftarrow r + cx$	$\xrightarrow{z}$	

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

<u>Completeness</u>: if $z = r + cx$, then

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

zero knowledge only required to hold against an honest verifier (e.g., view of the honest verifier can be simulated)

<u>Honest-Verifier Zero-Knowledge</u>: build a simulator as follows (familiar strategy: run the protocol in "reverse"):

on input (g, h):

1. Sample $z \xleftarrow{R} \mathbb{Z}_p$
2. Sample $c \xleftarrow{R} \mathbb{Z}_p$
3. set $u = g^z / h^c$ and output (u, c, z)

uniformly random challenge

uniformly random group element since z is uniformly random

chosen so that $g^z = u \cdot h^c$ (relation satisfied by a valid proof)

Simulated transcript is identically distributed as the <u>real</u> transcript with an <u>honest</u> verifier

What goes wrong if the challenge is not sampled uniformly at random (i.e., if the verifier is dishonest)

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

↳ To get general zero knowledge, we require that the verifier first <u>commit</u> to its challenge (using a <u>statistically</u> hiding commitment)

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

for simplicity, we assume P^* succeeds with probability 1

1. Run the prover P^* to obtain an initial message u.
2. Send a challenge $c_1 \xleftarrow{R} \mathbb{Z}_p$ to P^*. The prover replies with a response z_1.
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 $c_2 \xleftarrow{R} \mathbb{Z}_p$ to P^*. Let z_2 be the response of P^*.
4. Compute and output $x = (z_1 - z_2)(c_1 - c_2)^{-1} \in \mathbb{Z}_p$.

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

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

$$\Rightarrow \frac{g^{z_1}}{h^{c_1}} = \frac{g^{z_2}}{h^{c_2}} \Rightarrow g^{z_1 + c_2 x} = g^{z_2 + c_1 x}$$

$$\Rightarrow x = (z_1 - z_2)(c_1 - c_2)^{-1} \in \mathbb{Z}_p$$

with overwhelming probability, $c_1 \neq c_2$

Thus, extractor succeeds with <u>overwhelming</u> probability.

(Boneh-Shoup, Lemma 19.2)

If P^* succeeds with probability ε, then need to rely on "Rewinding Lemma" to argue that extractor obtains two accepting transcripts with probability at least $\varepsilon^2 - 1/p$.

The ability to extract a witness from any two accepting transcripts is very useful

↳ called <u>special soundness</u> (for 3-message protocols)

given (u, t_1, z_1) and $(u, t_2, z_2) \Rightarrow$ can extract the witness

initial message

challenge

response

[same initial message, different challenges]

3-message protocols that satisfy completeness, special soundness, and HVZK are called Σ-protocols

↳ Σ-protocols are useful for building signatures and identification protocols

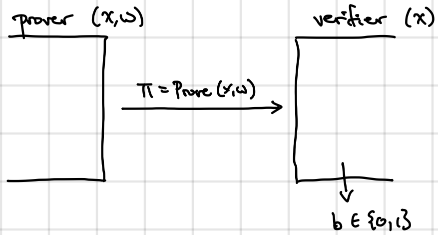
How can a prover both <u>prove knowledge</u> and yet be zero-knowledge at the same time?

↳ Extractor operates by "rewinding" the prover (if the prover has good success probability, it can answer most challenges correctly.

↳ But in the real (actual) protocol, verifier <u>cannot</u> rewind (i.e., verifier only sees prover on fresh protocol executions), which can provide zero-knowledge.

Many extensions of Schnorr's protocol to prove relations in the exponent.

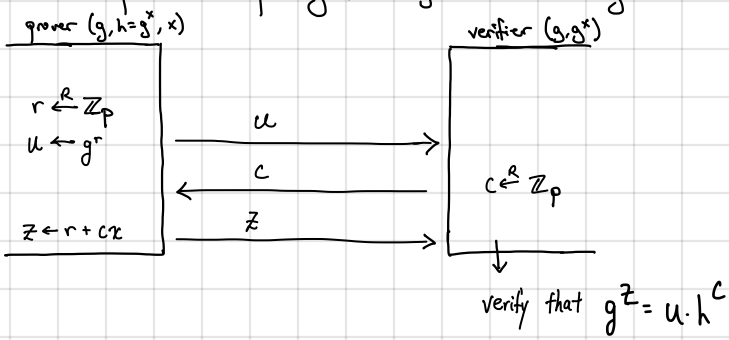
<u>Non-interactive zero-knowledge</u> (NIZK): Can we construct a zero-knowledge proof system where the proof is a <u>single</u> message from the prover to the verifier?



NIZKs for NP unlikely to exist for NP (unless NP $\subseteq$ BPP), but possible in the random oracle model (as well as in the common reference string model)

<u>Fiat-Shamir heuristic</u>: NIZKs in random oracle model

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



In this protocol, verifier's message is uniformly random (and in fact, is "public coin" — the verifier has <u>no</u> secrets)

<u>Key idea</u>: 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!

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

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

  $u \leftarrow g^r$ $c \leftarrow H(g, h, u, m)$ $z \leftarrow r + cx$

  $\sigma = (u, z)$

  } signature is a NIZK proof of knowledge of discrete log of h (with challenge derived from the message m)

- Verify(vk, m, σ): write $\sigma = (u, z)$, $\text{vk} = h$, 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 <u>proof of knowledge</u> of the discrete log (can be <u>extracted</u> from adversary)

Length of Schnorr's signature: vk: $(g, h = g^x)$ $\quad$ $\sigma: (g^r, \underbrace{c = H(g, h, g^r, m)}_{\text{can be computed given other components, so do not need to include}}, z = r + cx)$ $\quad$ Verification checks that $g^z = g^r h^c$

sk: x

$\Rightarrow |\sigma| = 2 \cdot |\mathbb{G}|$ $\quad$ [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 <u>384 bits</u>

128 bit challenge
+
256 bit group element

<u>Important note</u>: Schnorr signatures are <u>randomized</u>, and security relies on having <u>good</u> randomness

↳ What happens if randomness is reused for two different signatures?

Then, we have

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

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 $\sigma \leftarrow \mathrm{Sign}(sk, m; r)$.

↳ 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 $\quad$ [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: $(g, h = g^x)$ $\quad$ sk: x

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

  $u \leftarrow g^\alpha$ $\quad$ $r \leftarrow f(u) \in \mathbb{Z}_p$ $\quad$ 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 $r = f(u)$

  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)