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

↳ In most applications, we additionally require zero-knowledge

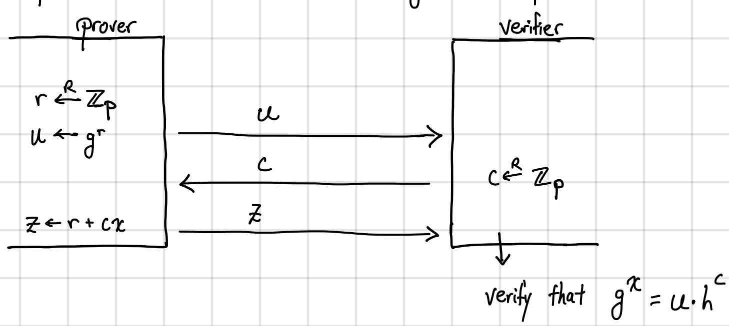
Note: knowledge is a strictly stronger property than soundness

↳ 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)



Completeness: 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)

Honest-Verifier Zero-Knowledge: 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 $c \xleftarrow{R} \mathbb{Z}_p$ — uniformly random challenge
3. set $u = g^z / h^c$ and output (u, c, z)

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 real transcript with an honest 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 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:

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 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

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

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

with overwhelming probability, $c_1 \neq c_2$

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

Thus, extractor succeeds with overwhelming 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 $\varepsilon^2 - 1/p$.

How can a prover both prove knowledge 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 cannot rewind (i.e., verifier only sees prover on fresh protocol executions), which can provide zero-knowledge.

Identification protocol from discrete log:

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

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

client → server

client ← server

client → server

protocol is precisely 3-round Schnorr proof of knowledge of discrete log

Essentially, the discrete log of h (base g) 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

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 HVZK)

More general view: Σ-protocols (Sigma protocols)

$g, h = g^x$

prover (x) — verifier

$\xrightarrow{g^r}$ "commitment"

$\xleftarrow{c}$ "challenge" (random string, "public-coin")

$\xrightarrow{r + cx}$ "response"

verifier has no secret randomness (Arthur-Merlin proofs)

protocol flow resembles a Σ

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

Properties: 1. Completeness
2. Honest-Verifier Zero-Knowledge
3. Proof of Knowledge

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

- Common discrete log: x such that $h_1 = g_1^x$ and $h_2 = g_2^x$ (useful for building a verifiable random function)
- DDH tuple: (g, u, v, w) is a DDH tuple — namely, that $u = g^\alpha$, $v = g^\beta$, and $w = g^{\alpha\beta}$ for $\alpha, \beta \in \mathbb{Z}_p$
  - ↳ useful for proving relations on ElGamal ciphertexts (e.g., that a particular ElGamal ciphertext encrypts either 0 or 1)

<u>Basic approach for electronic voting:</u>

$P_1 \xrightarrow{Enc(pk, x_1)}$ vote aggregator

$P_2 \xrightarrow{Enc(pk, x_2)}$ vote aggregator

$\vdots$

$P_n \xrightarrow{Enc(pk, x_n)}$ vote aggregator

only knows pk

vote aggregator $\xrightarrow{Enc(pk, \sum x_i)}$ voting authority

knows sk

↓

decrypt to learn $\sum x_i$: candidate 0 wins if sum $< \frac{n}{2}$; candidate 1 wins if sum $> \frac{n}{2}$

Assume two candidates (0/1)

<u>Requirement 1</u>: Public-key encryption scheme needs to be "additively homomorphic"

True for "exponential ElGamal"

<u>Setup</u>: Let $\mathbb{G}$ be group of order p and generator g

$x \xleftarrow{R} \mathbb{Z}_p$ $\quad$ pk: $(g, h = g^x)$

$\quad$ sk: x

<u>Encrypt$(pk, \underbrace{x}_{\in \mathbb{Z}_p})$</u>: $r \xleftarrow{R} \mathbb{Z}_p$

ct: $(g^r, h^r \cdot g^x)$

<u>Decrypt$(\underbrace{sk}_{=x}, \underbrace{ct}_{=(u,v)})$</u>: compute $z = \frac{v}{u^x}$ and output $m' \in \mathbb{Z}_p$ such that $g^{m'} = z$

this is solving discrete log in $\mathbb{G}$

[possible as long as m' is <u>small</u> — try every value of m' in the interval]

Given two ciphertexts $ct_0 = (g^{r_0}, h^{r_0} \cdot g^{x_0})$, $ct_1 = (g^{r_1}, h^{r_1} \cdot g^{x_1})$ ⇝ compute $(g^{r_0 + r_1}, h^{r_0 + r_1} g^{x_0 + x_1})$

↳ encryption of the sum $x_0 + x_1 \in \mathbb{Z}_p$

[can be used to sum encrypted votes; resulting value between 1 and n]

Basic voting protocol still <u>not</u> secure! Voter can be malicious and encrypt a non-0/1 value (e.g., −100 or 100)!

- Voters must prove that their vote is valid (i.e., encryption of 0/1), but <u>without</u> revealing the vote
- Language of valid ciphertexts (defined with respect to g, h)

$$\mathcal{L} = \{(u, v) \in \mathbb{G} : \exists r \in \mathbb{Z}_p : (u = g^r, v = h^r \text{ or } u = g^r, v = h^r \cdot g)\}$$

[Chaum-Pedersen]

Implies proof of knowledge of DDH tuples: if (g, u, v, w) is DDH tuple, then $v = g^r$, $w = u^r$ for some $r \in \mathbb{Z}_p$, so proving knowledge of common discrete log suffices