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

\[ \Rightarrow \text{In most applications, we additionally require zero-knowledge.} \]

Note: knowledge is a strictly stronger property than soundness.

\[ \Rightarrow \text{If protocol has knowledge error } e \Rightarrow \text{it also has soundness error } E \text{ (i.e., a dishonest prover convinces an honest verifier of a false statement with probability at most } e) \]

Proving knowledge of discrete log (Schnorr's protocol)

Suppose prover wants to prove it knows \( x \) such that \( h = g^x \) (i.e., prover demonstrates knowledge of discrete log of \( h \) base \( g \))

\[ \text{prover} \]
\[ \begin{array}{cccc}
 r & \in & Z_p & \quad \Rightarrow \quad \text{uniformly random challenge}

e & \in & Z_p & \\
 z & = & r & + c z \\
 \end{array} \]

\[ \text{ verifier} \]
\[ \begin{array}{cccc}
 u & \in & g^r & \\
 c & \in & Z_p & \\
 z & \in & Z_p & \\
 \end{array} \]

\[ u = g^r \]
\[ c z = u h \]

Completeness: if \( z = r + c z \), then
\[ g^z = g^r g^{c z} = g^r c z = u h \]

\[ g^2 = u_h \]

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 \( z \in Z_p \)
2. Sample \( c \in Z_p \)
3. Set \( u = g^{z/c} \) and output \((u, c, z)\)

Simulation is identical to 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)

\[ \Rightarrow \text{To get general zero-knowledge, we require that the verifier first commit to its challenge (using a statistically hiding commitment)} \]

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

1. Run the prover \( P^* \) to obtain an initial message \( u \).
2. Send a challenge \( c_1 \in Z_p \) 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 \( c_2 \in Z_p \) to \( P^* \). Let \( z_2 \) be the response of \( P^* \).
4. Compute and output \( x = (z_1 - z_2)(c_1 - c_2)^{-1} \in Z_p \).
Since $P^*$ succeeds with probability 1 and the extractor perfectly simulates the honest prover's behavior, with probability 1, both $(u, c_1, z_1)$ and $(u, c_2, z_2)$ are both accepting transcripts. This means that
\[
\frac{g^{z_1}}{U \cdot h_{c_1}} = \frac{g^{z_2}}{U \cdot h_{c_2}} \quad \Rightarrow \quad \frac{g^{z_1}}{h_{c_1}} = \frac{g^{z_2}}{h_{c_2}} \quad \Rightarrow \quad g^{z_1 + c_1 x} = g^{z_2 + c_2 x}
\]

\[
\Rightarrow \quad x = (z_1 - z_2) (c_1 - c_2)^{-1} \in \mathbb{Z}_p \quad c_1 \neq c_2
\]

Thus, extractor succeeds with overwhelming probability.

(Boneh-Shoup, lemma 19.2)

If $P^*$ succeeds with probability $\epsilon$, then need to rely on "Rewinding lemma" to argue that extractor obtains two accepting transcripts with probability at least $\epsilon^2 - \frac{1}{p}$.

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 cannot provide zero-knowledge.

Identification protocol from discrete log:

\[
\text{Client} \quad (x) \quad \text{Server} \quad (y, h = g^y) \quad \text{Public verification key}
\]

\[
\xrightarrow{\text{protocol is precisely 3-round}} \quad \text{Schnorr proof of knowledge of discrete log}
\]

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: $\Sigma$-protocols (Sigma protocols)

\[
g, h = g^y \quad \text{prover} \quad (x) \quad \text{verifier} \quad g^r \quad \text{"challenge"} \quad r + c \cdot x \quad \text{"response"}
\]


Protocols with this structure (commitment-challenge-response) are called $\Sigma$-protocols (Sigma protocols).
Many variants of Schnorr protocols can be used to prove knowledge of statements like:

- **Common discrete log**: \( \lambda \) such that \( h_1 = g^{\lambda} \) and \( h_2 = g^{\lambda^2} \) (useful for building a verifiable random function.)
- **DDH tuple**: \((g, u, v, w)\) is a DDH tuple – namely, that \( u = g^a, v = g^b, \) and \( w = g^{ab} \) for \( a, b \in \mathbf{Z}_p \)

\( \implies \) useful for proving relations on ElGamal ciphertexts (e.g., that a particular ElGamal ciphertext encrypts either 0 or 1)

**Basic approach for electronic voting:**

\[
\begin{array}{c}
P_1 \xrightarrow{\text{Exc}(ph, x_1)} \text{vote} \xrightarrow{\text{Exc}(ph, \Sigma x_i)} \text{voting authority} \\
P_2 \xrightarrow{\text{Exc}(ph, x_2)} \text{vote} \xrightarrow{\text{Exc}(ph, \Sigma x_i)} \text{voting authority} \\
\vdots \\
P_n \xrightarrow{\text{Exc}(ph, x_n)} \text{vote} \xrightarrow{\text{Exc}(ph, \Sigma x_i)} \text{voting authority}
\end{array}
\]

**Assume two candidates (0/1)**

**Requirement 1:** Public-key encryption scheme needs to be “additively homomorphic”

- True for “exponential ElGamal”

**Setup:** Let \( G \) be group of order \( p \) and generator \( g \)

\[
\begin{array}{c}
\lambda \in \mathbf{Z}_p \quad \text{pk}: (g, h = g^\lambda) \\
\lambda \in \mathbf{Z}_p \quad \text{sk}: \lambda
\end{array}
\]

**Encrypt(pk, x):**

\[
r \in \mathbf{Z}_p \quad \text{ct} : (g^r, h^r \cdot g^x)
\]

**Decrypt(sk, ct):**

\[
\text{compute } z = \frac{v}{w^\lambda} \text{ and output } m' \in \mathbf{Z}_p \text{ such that } g^{m'} = z \text{ every value of } m' \text{ in the interval}
\]

Given two ciphertexts \( c_{t_0} = (g^{r_0}, h^{r_0} \cdot g^{x_0}) \rightarrow \) compute \( (g^{r_0 + r_1}, h^{r_0 + r_1} \cdot g^{x_0 + x_1}) \)

\( \rightarrow \) encryption of the sum \( x_0 + x_1 \in \mathbf{Z}_p \)

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

Basic voting protocol still not 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 without revealing the vote

- Language of valid ciphertexts (defined with respect to \( (g, h) \))

\[
L = \{(u, v) \in G : \exists r \in \mathbf{Z}_p \mid u = g^r, v = h^r \text{ or } u = g^r, v = h^r \cdot g \}
\]

[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 \( r \in \mathbf{Z}_p \), so proving knowledge of common discrete log suffices.