Identification protocol from discrete log:

\[ \text{client's secret (ecdential)} \rightarrow \text{public verification key} \]

\[ \text{client (x)} \rightarrow \text{server (g, h=g^x)} \]

\[ \text{protocol is precisely 3-round} \]

\[ \text{Schnorr proof of knowledge of discrete log} \]

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
(Achieve) 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)

\[
\begin{align*}
\text{prover (x)} & \rightarrow \text{verifier} \\
g^x & \rightarrow \text{"commitment"} \\
 & \leftarrow \text{"challenge"} \\
1 & \text{"response"} \\
1 & \text{"response"} \\
1 & \text{"response"}
\end{align*}
\]

Verifier has no secret randomness (Arthur-Merlin proofs)

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

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: \( x \) such that \( h_1 = g^x \) and \( h_2 = g^x \) (useful for building a verifiable random function).
- DDH tuple: \( (g, u, v, w) \) is a DDH tuple - namely, that \( u = g^x \), \( v = g^y \), and \( w = g^{x+y} \) for \( x, y \in \mathbb{Z}_p \).
⇒ Useful for proving relations on ElGamal ciphertexts (e.g., that a particular ElGamal ciphertext encrypts either 0 or 1).

Basic approach for electronic voting:

\[
\begin{align*}
P_1 & \xrightarrow{\text{Enc}(pk, x_i)} \text{vote} \\
P_2 & \xrightarrow{\text{Enc}(pk, x_i)} \text{aggregator} \\
& \text{Enc}(pk, \Sigma x_i) \text{voting authority} \\
P_n & \xrightarrow{\text{Enc}(pk, x_i)} \\
\end{align*}
\]

Assume two candidates (0/1)

\[ \text{Requirement 1: Public-key encryption scheme needs to be "additively homomorphic"} \]

\[ \text{True for "exponential ElGamal"} \]
Setup: Let $G$ be group of order $p$ and generator $g$
\[
x \in \mathbb{Z}_p \quad qk: (g, h = g^x) \quad sk: x
\]

Encrypt $(pk, x)$:
\[
\begin{aligned}
& c_t \in \mathbb{G}_1 \quad \text{ct} : (g^r, h^r \cdot g^x) \\
& r \in \mathbb{Z}_p
\end{aligned}
\]

Decrypt $(sk, ct)$:
\[
\begin{aligned}
& z = \frac{v}{u^x} \quad \text{compute } z
\end{aligned}
\]

Given two ciphertexts $ct_0 = (g^{r_0}, h^{r_0} \cdot g^{x_0})$
\[
ct_i = (g^{r_i}, h^{r_i} \cdot g^{x_i})
\]

\[
\Rightarrow \quad \text{encryption of the sum } x_0 + x_i \in \mathbb{Z}_p
\]

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

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

Starting point: proof of knowledge of two discrete logs (for fixed $g_1, g_2$)
\[
L = \{(u, v) \in G : \exists r \in \mathbb{Z}_p : u = g^r, v = g^r\}
\]

Verifier
\[
\begin{aligned}
& c_t \in \mathbb{G}_1 \\
& u_1 = g^{r_1} \\
& u_2 = g^{r_2} \\
& c = \frac{v}{u_1^x} \\
& z = r + c x
\end{aligned}
\]

Check that $g_z = u_1 \cdot h_z^c$ and $g_z^2 = u_2 \cdot h_z^c$

Completeness and HVZK follows as in Schnorr’s protocol.

Knowledge: Two scenarios:

1. If prover uses inconsistent commitment (i.e., $u_1 = g^{r_1}$ and $u_2 = g^{r_2}$ where $r_1 \neq r_2$), then over choice of honest verifier’s randomness, then prover can only succeed with probability at most $\frac{1}{p}$:
\[
\begin{aligned}
& z = r_1 + x_c r = r_1 + x_c c \quad \text{(if verifier accepts)} \\
& u_1 = g^{r_1} \\
& h_1 = g^{r_1} \\
& u_2 = g^{r_2} \\
& h_2 = g^{r_2}
\end{aligned}
\]

This means that
\[
(r_1 - r_2) = c (x_c - x)
\]

If $r_1 \neq r_2$, there is at most 1 $c \in \mathbb{Z}_p$ where this relation holds. Since $c$ is uniform over $\mathbb{Z}_p$, the verifier accepts with probability at most $\frac{1}{p}$.

2. If prover succeeds with probability, then it must use a “consistent” commitment. Can build extractor just as in Schnorr’s protocol. Knowledge error larger by additive $\frac{1}{p}$ term (from above analysis).
Our language of valid notes:
\[ L = \{ (u,v) \mid \exists r : (u=g^r, v=g^r \text{ or } u=g^{2r}, v=g^r g^r) \} \]
Equivalently: either know \( r \) such that \( u=g^r, v=g^r \) or \( u=g^{2r}, v=g^r g^r \).

Looks like statement for knowledge of two discrete logs
(either for statement \((u,v)\) or for statement \((u,g^r g^r)\))

**Or-proof:** A general approach for proving or of two statements (without revealing which one is true)
We will illustrate for simple case of
\[ L = \{ (h_1, h_2) \mid \exists x (h_1=g^x \text{ or } h_2=g^{2x}) \} \] [for fixed generator \( g \)]

Prover demonstrates knowledge of discrete log of either \( h_1 \) or \( h_2 \)

Starting point: Run two copies of Schnorr to prove knowledge of \((x_1,x_2)\) such that \( h_1=g^{x_1} \) and \( h_2=g^{x_2} \)

\[
\begin{align*}
\text{Prover:} & \quad r_1, r_2 \in Z_p \\
& \quad u_1=g^{r_1}, u_2=g^{r_2} \\
\text{Verifier:} & \quad c_1, c_2 \\
& \quad z_1 = c_1 r_1 + r_2, z_2 = c_1 r_2 + r_1 \\
& \quad c_1, c_2 \in Z_p \\
& \quad z_1, z_2 \\
\end{align*}
\]

**Problem:** Honest prover only knows one of \( x_1 \) or \( x_2 \). So it cannot correctly answer both challenges (unless it knew both \( x_1 \) and \( x_2 \)).

Key idea: Prover will simulate the transcript it does not know.

Suppose prover knows \( x=x_1 \). Then, it will first run the Schnorr simulator on input \((g,x)\) to obtain transcript \((z_1, c_2, z_2)\).

But challenge \( c_2 \) may not match \( z_1 \). To address this, we will have the verifier send a single challenge \( c \in Z_p \) and the prover can pick \( c_1 \) and \( c_2 \) such that \( c_1 + c_2 = c \in Z_p \)

\[
\begin{align*}
\text{Prover (} x_1 \text{)} & \quad (u_1, c_1, z_1, z_2) \leftarrow S(g^x) \\
& \quad r_1 \in Z_p \\
& \quad u_1 = g^{r_1} \\
& \quad c_1 = c - c_1, z_1 = c_1 r_1 + r_1 \\
& \quad z_2 = r_1 + c_1 x_1 \\
& \quad c_2 = (z_2 - c_1) / z_1 \\
& \quad z_2 = r_1 + c_1 x_1 \\
& \quad \text{check that} \\
& \quad z_1 = u_1 c_1, z_2 = u_1 c_2 \\
\end{align*}
\]

Completeness, HVZK and proof of knowledge follow very similarly as in the proof of Schnorr’s protocol.

Proving that \((u,v)\) have the form \((g^x,h^y)\) or \((u,g^r g^r)\) can be done by combining or-proof with proof of knowledge of two discrete logs described above.

- Narrowly, prover simulates proof of instance that is false and proves the statement that is true.