CS 6501 Week 8: Zero-Knowledge Proof Systems

In a zero-knowledge proof system, a prover can convince the verifier that some statement x is true (without revealing anything more about x).

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 $\mathcal{L}$)

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

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 ε $\Rightarrow$ 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

$r \xleftarrow{R} \mathbb{Z}_p$

$u \leftarrow g^r$

$\xrightarrow{\quad u \quad}$

verifier: $c \xleftarrow{R} \mathbb{Z}_p$

$\xleftarrow{\quad c \quad}$

prover: $z \leftarrow r + cx$

$\xrightarrow{\quad z \quad}$

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

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 $c \xleftarrow{R} \mathbb{Z}_p$
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 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

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

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

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

with overwhelming probability, $c_1 \neq c_2$

Thus, extractor succeeds with 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 <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.

<u>Identification Protocol from Discrete Log</u>

Suppose a client wants to authenticate to the server

↳ <u>Goal</u>: security against <u>active</u> adversaries (adversary sees contents of the server and can interact arbitrarily with the client)

vanilla password-based authentication does not provide active security in this setting

Can directly build such a scheme from Schnorr's protocol:

client's secret (credential) — public verification key

client (x) $\qquad$ server $(g, h = g^x)$

$\longrightarrow$

$\longleftarrow$

$\longrightarrow$

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 <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

↳ 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) $\qquad$ verifier

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

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

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

verifier has <u>no</u> secret randomness (Arthur-Merlin proofs)

protocol flow resembles a Σ

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

<u>Properties</u>: 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)
  - ↳ useful building block in constructions of DDH-based oblivious transfer (OT) protocols — Naor-Pinkas (more details next lecture)

↳ Reduces to proving common discrete log: (g, u, v, w) is a DDH tuple if and only if there is an x such that $v = g^x$ and $w = u^x$

Showing that $h_1 = g_1^x$ and $h_2 = g_2^x$:

prover		verifier
$r \xleftarrow{R} \mathbb{Z}_p$	$u_1 = g_1^r$, $u_2 = g_2^r$ $\longrightarrow$	
	$\longleftarrow$ c	$c \xleftarrow{R} \mathbb{Z}_p$
	$z = r + cx$ $\longrightarrow$	

Check that $g_1^z = u_1 \cdot h_1^c$ and $g_2^z = u_2 \cdot h_2^c$

Completeness and HVZK follows as in Schnorr's protocol.

Knowledge: Two scenarios:

1. If prover uses inconsistent commitment (i.e., $u_1 = g_1^{r_1}$ and $u_2 = g_2^{r_2}$ where $r_1 \neq r_2$), then over choice of honest verifier's randomness, then prover can only succeed with probability at most $1/p$:

$$z = r_1 + x_1 c = r_2 + x_2 c \quad \text{(if verifier accepts)}$$

$u_1 = g_1^{r_1}$ $h_1 = g_1^{x_1}$ $u_2 = g_2^{r_2}$ $h_2 = g_2^{x_2}$

This means that

$$(r_1 - r_2) = t(x_2 - x_1)$$

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 $1/p$

2. If prover succeeds with $1/\text{poly}(\lambda)$ probability, then it must use a "consistent" commitment. Can build extractor just as in Schnorr's protocol. Knowledge error larger by additive $1/p$ term (from above analysis).

If we want to prove the AND of many statements, then we can prove each one in sequence. What if we want to prove the OR of many statements. The difficulty is not revealing which statement is true (or in the case of proof of knowledge, which witness the prover knows).

We will work with the following: Prover wants to show that it knows either x_1 or x_2 such that $h_1 = g^{x_1}$ or $h_2 = g^{x_2}$

"1 out of 2 discrete logs"

↳ Statement: (g, h_1, h_2)

Witness: x_1 or x_2 where $h_1 = g^{x_1}$, $h_2 = g^{x_2}$

Starting point: Run Schnorr protocol in parallel:

prover		verifier
$r_1, r_2 \xleftarrow{R} \mathbb{Z}_p$	$u_1 = g^{r_1}$ $u_2 = g^{r_2}$ $\longrightarrow$	
	$\longleftarrow$ c_1, c_2	$c_1, c_2 \xleftarrow{R} \mathbb{Z}_p$
	$z_1 = r_1 + c_1 x_1$, $z_2 = r_2 + c_2 x_2$ $\longrightarrow$	

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_1. Then, it will first run the Schnorr simulator on input (g, h_2) to obtain transcript $(\tilde{u}_2, \tilde{c}_2, \tilde{z}_2)$.

↳ But challenge c_2 may not match $\tilde{c}_2$... To address this, we will have the verifier send a single challenge $c \xleftarrow{R} \mathbb{Z}_p$ and the prover can pick c_1 and c_2 such that $c_1 + c_2 = c \in \mathbb{Z}_p$

prover (x_1) | verifier

$(\tilde{u}_2, \tilde{c}_2, \tilde{z}_2) \leftarrow S(g, h_2)$

$r_1 \xleftarrow{R} \mathbb{Z}_p$

$\xrightarrow{\quad u_1 \quad \tilde{u}_2 \quad}$

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

$c_1 \leftarrow c - \tilde{c}_2$

$z_1 \leftarrow r_1 + c_1 x_1$

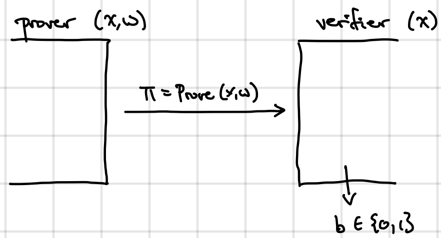
$\xrightarrow{\quad c_1 \quad z_1 \quad \tilde{z}_2 \quad}$

check that

$$g^{z_1} = u_1 h_1^{c_1}$$
$$g^{\tilde{z}_2} = \tilde{u}_2 h_2^{c - c_1}$$

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

<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?



Why do we care? <u>Interaction</u> in practice is <u>expensive</u>!

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

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

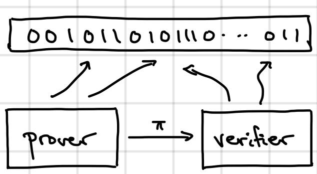
if $x \in \mathcal{L}$: $S(x) \to \pi$ and π should be accepted by the verifier (by ZK)

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:

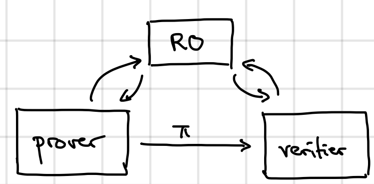
Common random/reference string (CRS) model:



prover and verifier have access to shared randomness (could be a <u>uniformly</u> random string or a <u>structured</u> 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:

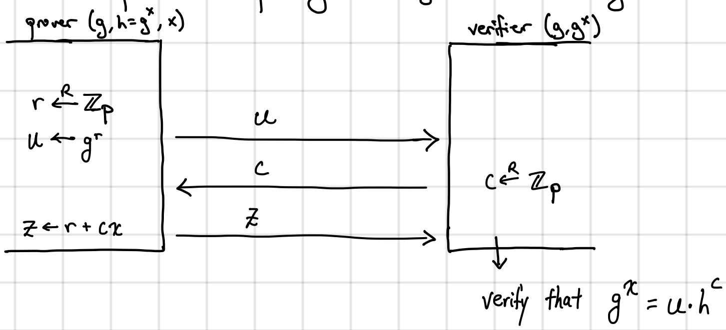


in this model, simulator can "program" the random oracle (again, asymmetry between real prover and the simulator)

$\Longrightarrow$ In both cases, simulator has additional "power" than the real prover, which is critical for enabling NIZK constructions for NP.

Fiat-Shamir heuristic: from Σ-protocols to NIZK in RO 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 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 $c \leftarrow H(g, h, u)$. $\leftarrow$ prover can now compute this quantity on its own!

Security of Fiat-Shamir:

1. **Completeness**: Same as Schnorr's protocol
2. **Zero-Knowledge**: Same as in Schnorr's protocol; namely, simulator samples $c \xleftarrow{R} \mathbb{Z}_p$, $z \xleftarrow{R} \mathbb{Z}_p$, computes u, and then programs RO at (g, h, u) to c.
3. **Knowledge**: Construct extractor as follows: given (possibly malicious) prover P^*:
   1. Run P^* to obtain proof $\pi = (u, z)$ where challenge $c = H(g,h,u)$ at verification time
      ↳ Note that at some point, P^* must have queried the random oracle on input (g, h, u)
   2. Run P^* again, but when it queries RO, use different responses
      ↳ Can extract discrete log as before

   need to argue that with high probability, P^* will output proof with same committed value u (follows by "forking lemma")

Signatures from discrete log in RO model (Schnorr):

- Verification key is $(g, h = g^x)$ and signing key is x
- To sign a message m, signer proves knowledge of x (discrete log of h) using Fiat-Shamir (and where challenge is derived from message): e.g., $c \leftarrow H(g, h, u, m)$.
- Security essentially follows from security of Schnorr's identification protocol (together with Fiat-Shamir)
  ↳ specifically, challenger answers signing queries using the ZK simulator (programming RO as needed for consistency)
  ↳ 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)

More generally, any Σ-protocol can be used to build a signature scheme using the Fiat-Shamir heuristic (by using the message to derive the challenge via RO)

Length of Schnorr's signature: vk: $(g, h = g^x)$ sk: x

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

$\Longrightarrow |\sigma| = 2 \cdot |\mathbb{G}|$ [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

In practice, we use a variant of Schnorr's signature scheme called $\underbrace{\text{DSA}}_{}$ / ECDSA

TLS protocol

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]

<u>Important note</u>: Schnorr signatures (and DSA / ECDSA) 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

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

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

↳ in PlayStation 3, the randomness was a fixed <u>constant</u>! Enables hackers to deliver <u>arbitrary</u> firmware updates to device!

and in some <u>bad</u> Bitcoin wallets

<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.