Zero Knowledge Part II

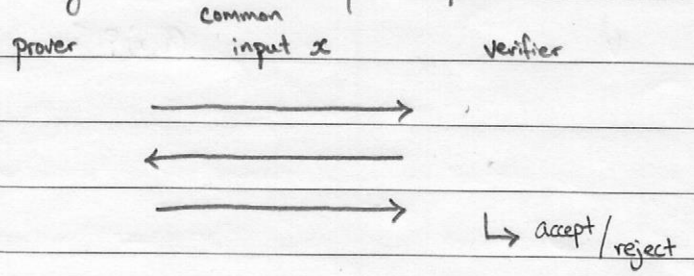
Logistics : PS2 due now

Project milestone due next week (May 17th)

- 1-2 pages using provided template
- Should roughly coincide with first half of project (summarize results thus far)

Previous lecture : Notion of "zero-knowledge" – one of the most fundamental ideas in modern cryptography

- Convince someone that a statement is true without revealing anything else about the statement
- Recall the setting of an interactive proof system :



completeness : true statements can be proven

soundness : false statements will be rejected

NP : the class of languages with non-interactive proofs

IP : the class of languages with interactive proofs [IP = PSPACE]

- Zero-knowledge : proofs reveal no more information about statement
  - ↳ captured by the notion of a simulator :

$$\forall \text{ verifiers } V^*, \ \exists \text{ efficient simulator } S, \ \forall x \in \mathcal{L} :$$

$$\text{View}_{V^*}\left[P(x) \leftrightarrow V^*(x)\right] \overset{c}{\approx} \text{Sim}(x)$$

  - ↳ whatever V^* could have learned by interacting with the simulator, it could also learn by running the simulator (which only knows x)
- Zero knowledge proofs possible for all of NP (in fact, PSPACE)
  - ↳ for NP, show via reduction to 3-coloring
  - ↳ relies on use of commitment scheme ("sealed envelopes")
  - ↳ rough sketch :

3-round ZK with constant soundness

1. Prover permutes the colors and commits to coloring
2. Verifier challenges prover on an edge
3. Prover opens commitment and verifier checks consistency

Proofs of Knowledge

So far: we can demonstrate that statements are true without revealing anything more
e.g., can show that a graph G is 3-colorable without revealing the coloring

But in many cases, we want a stronger property: we want to be convinced that the prover "knows" a valid witness (e.g., verifier convinced prover knows why statement is true)

- Proving knowledge of discrete log

$$\mathcal{L} = \{ h = g^x \text{ for some } x \in \mathbb{Z}_q \} = \mathbb{G}$$

In this case, all statements are true (if g is a generator), so zero knowledge proofs of membership are trivial. But very useful to define zero-knowledge proof of knowledge (ZKPoK) of discrete log. Basis of identification protocols.

What does it mean to "know" something? [GMR85]

- If a prover (possibly malicious) is able to convince the verifier to accept, then it should be possible to "extract" a witness from the prover
- In other words, a prover only succeeds if it "knows" a witness for the statement being proved
- Captured by defining the notion of an extraction algorithm

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

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

could also be polynomially smaller

knowledge error

Intuition: If prover is able to convince verifier, then prover can be leveraged to extract the witness w. Thus, prover must know some witness w (in fact, prover can always run the extractor itself to obtain witness).

Trivial proof of knowledge: simply send witness in the clear $\Rightarrow$ combine with

Proving Knowledge of Discrete Log (Schnorr)

Suppose prover wants to demonstrate that it knows x such that $h = g^x$ where, g, h publi

$$
\begin{array}{lcl}
\underline{\text{prover}} & & \underline{\text{verifier}} \\
r \xleftarrow{R} \mathbb{Z}_q & & \\
u \leftarrow g^r & \xrightarrow{\quad u \quad} & \\
 & \xleftarrow{\quad c \quad} & c \xleftarrow{R} \mathbb{Z}_q \\
z \leftarrow r + cx & \xrightarrow{\quad z \quad} & \text{verify that } g^z = u \cdot h^c
\end{array}
$$

Completeness: if $z = r + cx$, then $g^z = g^{r+cx} = g^r g^{cx} = u \cdot h^c$

Honest-Verifier Zero Knowledge: construct simulator as follows: (run protocol backwards)

on input (g, h):

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

sampled uniformly at random

satisfies verification relation

uniformly distributed since z is uniform

hat happens if
is not uniform?
annot simulate by
ampling random z!
ndeed, can leak information about x)

How to get general zero-knowledge: high-level idea is to require verifier to first commit to its challenge query in the first round of the protocol (have to be careful dealing with aborts — can use standard techniques here)

Proof of Knowledge (PoK): Construct extractor as follows:

- model prover $P^* = (P_1^*, P_2^*)$ for computing the first and second messages, respectively
- extractor operates as follows:
  1. $(u, st) \leftarrow P_1^*(x)$
  2. choose $c_1 \xleftarrow{R} \mathbb{Z}_q$ and $c_2 \xleftarrow{R} \mathbb{Z}_q$ and compute
     $z_1 \leftarrow P_2^*(st, c_1)$ and $z_2 \leftarrow P_2^*(st, c_2)$
  3. output $x = (z_1 - z_2)(c_1 - c_2)^{-1} \in \mathbb{Z}_q$
- suppose P^* convinces honest verifier with probability ε

$$\Rightarrow \left.\begin{array}{l}\text{with probability } \varepsilon,\ g^{z_1} = u \cdot h^{c_1} \\ \text{with probability } \varepsilon,\ g^{z_2} = u \cdot h^{c_2}\end{array}\right\} \text{ with probability } \varepsilon^2:$$

$z_1 = r + c_1 x$
and

Proving Knowledge of Discrete Log (schnorr)

- With probability ε^2, $z_1 - z_2 = x(c_1 - c_2)$ and the extractor outputs x
- Proof technique often referred to as "rewinding" trick because the extractor rewinds the execution state of the prover (force the prover to answer two challenges using the same randomness)

Meta-question: how is the proof zero-knowledge and yet, we can extract the witness?

Two different settings!
- Zero-knowledge property: relies on fresh invocation of protocol
- Proof of knowledge: rewind the prover to extract secrets (real execution cannot be rewound)

Identification Protocols from Discrete Log:

- Client wants to authenticate to the server
- Security against active adversaries (adversary sees contents of server, can interact with client) → goal is to impersonate client

$$\begin{tikzcd}
\text{client } (g, h = g^x, x) \\ (\text{prover}) \arrow[r] & \text{server } (g,h) \\ (\text{verifier}) \arrow[l] \\ \arrow[r] &
\end{tikzcd}$$

challenge-response protocol is precisely Schnorr's protocol

- Secure against active adversaries (with hardness reducing to discrete log)
  - Reduces to hardness of discrete log (adversary that successfully convinces server to accept must know the discrete log – proof of knowledge)
  - Interactions with the client can be entirely simulated (zero-knowledge)
    ↳ need general zero knowledge (rather than HVZK) if fully active adversary
- Compare with password-based authentication, secret-key challenge-response, and public-key challenge-response

Another Perspective: Sigma Protocols

prover $\qquad$ verifier

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

$$\xleftarrow{c} \quad \text{"challenge" (random string)}$$

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

Arthur-Merlin proofs

Sigma Protocols

- Protocols with this structure (3-round, public-coin, HVZK) are often called sigma protocols
  ↳ protocol flow (somewhat) resembles a Σ

- Many variants of Schnorr's protocol : can be used to prove knowledge of statements like
  - "1-out-of-2" discrete logs : x such that $h_1 = g^x$ or $h_2 = g^x$
  - DDH tuple : (g, x, y, z) is a DDH tuple : $x = g^a$, $y = g^b$, $z = g^{ab}$
    (Chaum-Pedersen protocol)

Chaum-Pedersen Protocol

$$(\underset{}{g}, \overset{g^a}{\overset{\shortparallel}{x}}, \overset{g^b}{\overset{\shortparallel}{y}}, \overset{g^c}{\overset{\shortparallel}{z}}) \quad \text{where } c = ab$$

→ amounts to checking that $y = g^b$ and $z = x^b$
↳ prove knowledge of two discrete logs and argue consistency

prover | **verifier**

two commitments for base g and base x — $s \xleftarrow{R} \mathbb{Z}_q$ $\quad \xrightarrow{\;V = g^s,\; w = x^s\;}$

single challenge for shared exponent — $\xleftarrow{\quad t \quad}$ $\quad t \xleftarrow{R} \mathbb{Z}_q$

verify each discrete log relationship — $\xrightarrow{\;u = s + bt\;}$ check $g^u = V \cdot y^t$, $x^u = w \cdot z^t$

- Completeness + HVZK just as in Schnorr protocol
- Soundness : 1. Let $V = g^s$ and $w = x^{s'}$ be prover's commitment (where s need not equal s')
  2. Prover needs to produce u such that

$$u = s + bt \quad \text{and} \quad u = a^{-1}(as' + ct)$$

$$\Rightarrow s + bt = s' + a^{-1}ct \Rightarrow t = (s - s')\underbrace{(a^{-1}c - b)^{-1}}_{\text{non-zero since } c \neq ab}$$

$\therefore$ only 1 possible value of t that prover can use to produce valid proof

$\Longrightarrow$ achieves soundness $1/q$

The Fiat-Shamir Heuristic (NIZKs)

- Oftentimes, interaction is <u>costly</u>: can we make our proofs non-interactive?
- Non-interactive proofs are transferable (multiple verifiers can independently check the proof)

- All of NP have non-interactive proofs (namely, the witness is the proof)
- But what about zero-knowledge (NIZKs)?
  - If $\mathcal{L}$ has a NIZK proof system, then $\mathcal{L} \in BPP$
  - <u>Proof (Sketch)</u>. On input a statement x, run the simulator on x to obtain a proof π. Output Verify(x, π). } BPP decis algorith

Why does this argument not apply for interactive ZK proofs?

<u>Completeness</u>: If $x \in \mathcal{L}$, then by (perfect) zero-knowledge, simulator outputs valid proof and completeness follows by completeness of the NIZK.

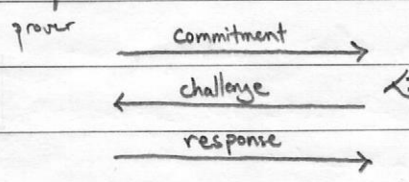
(perfect) → can be relaxed for computational zero knowledge

<u>Soundness</u>: If $x \notin \mathcal{L}$, then Verify(x, π) should reject with high probability (over the randomness used in the verification algorithm) regardless of π by soundness of the NIZK.

Since it is widely believed that $NP \not\subseteq BPP$, unlikely that NIZKs for NP exist in the standard model.

But we should not be deterred by impossibility results! Solution: work in the random oracle model

- Take a Σ-protocol:



⟸ just a random string (public-coin protocol)

↳ we can derive it from a random oracle

- Fiat-Shamir Heuristic

1. On a statement x, generate commitment r.
2. Derive challenge as $c \leftarrow H(x, r)$
3. Compute response t given challenge c.
4. Output proof as $\pi = (r, t)$

To verify: 1. Compute challenge $c \leftarrow H(x, r)$
2. Output proof $\pi = (r, t)$

More generally: apply to multi-round protocol

Completeness: Immediate by completeness of underlying scheme.

Soundness: Follows roughly from fact that challenges are uniformly distributed

Zero-Knowledge: Follows from zero-knowledge of underlying protocol (program the random oracle at query values appropriately)

<u>Main Application: Schnorr Signatures and DSA/ECDSA</u>

Recall Schnorr's ID protocol:

<u>prover</u> $\qquad h = g^x \qquad$ <u>verifier</u>

$$\xrightarrow{\quad g^r \quad}$$
$$\xleftarrow{\quad t \quad}$$
$$\xrightarrow{\quad rx + t \quad}$$

- Invoke Fiat-Shamir and derive challenge t as $H(g^x, g^r, m)$ where m is the message
- Yields a signature scheme based on discrete log (in random oracle model)
  - ↳ existentially unforgeable under discrete log (use fact that Schnorr is a proof of knowledge)
  - ↳ successful forger can be leveraged to break discrete log (by reprogramming the RO)
- Forms the basis of DSA/ECDSA (what is used on the web today) – much shorter than RSA signatures

- The big picture: started with a purely theoretic topic (what can we prove using interaction and what is the knowledge complexity of these proof systems) and obtained a highly practical, widely deployed cryptographic scheme!

- What else do we know?
  - Can also build NIZKs in the common random string model as well as common reference string (CRS) model (assume that prover and verifier have access to some shared string "in the sky")
    - NIZKs for NP can be built from trapdoor permutations (e.g., RSA) [Feige Lapidot Shamir] or pairings [Groth Sahai] in the CRS model
  - More exotic: proof systems where the proof is succinct (e.g., verification time + proof size is polylogarithmic in size of circuit / running time of the NP verifier) – SNARGs / SNARKs
    - Can construct from pairings or lattice-based assumptions
    - Has become the basis of Zerocash (some of the most sophisticated crypto to be deployed at scale!)