Interactive and Zero-Knowledge proofs

Subhashis Banerjee, Subodh Sharma

Department of Computer Science and Engineering
IIT Delhi

November 6, 2020

Knowledge

- To quantify the knowledge inherent in a message m, it is sufficient to quantify how much easier it becomes to compute some new function given m.
- Suppose Alice sends 0^n to Bob. Bob gains no new knowledge, because Bob could have produced the message himself.
- Suppose instead, that Alice sends Bob the message consisting of "the preimage of the preimage ... (n times) of 0 for a one-way function". That certainly would be new knowledge.

Knowledge

- The amount of knowledge conveyed in a message can be quantified by considering the running time and size of a Turing machine that generates the message.
- A message that can be generated by constant-sized Turing machine that runs in polynomial-time in n conveys no knowledge.
- For randomly selected messages: "Alice conveys zero knowledge to Bob if Bob can sample from a distribution of messages that is computationally indistinguishable from the distribution of messages that Alice would send."
- This is distinct from "information" and *Shannon entropy*. Messages that convey zero information may actually contain knowledge.

Example: Zero-Knowledge encryption

- A private-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a *Zero-Knowledge encryption scheme* if there exists a p.p.t. simulator algorithm $\mathcal{S}$ such that $\forall$ non uniform p.p.t. $\mathcal{D}$, $\exists$ a negligible function $\epsilon(n)$, such that $\forall m \in \{0,1\}^n$ it holds that $\mathcal{D}$ distinguishes the following distributions with probability at most $\epsilon(n)$:
  - $\{k \leftarrow \mathsf{Gen}(1^n) : \mathsf{Enc}_k(m)\}$
  - $\{\mathcal{S}(1^n)\}$
- If the above distributions are identical then it is *perfect Zero Knowledge*.
- A similar definition can be used for public-key encryption; $\mathcal{D}$ cannot distinguish between:
  - $\{p_k, s_k \leftarrow \mathsf{Gen}(1^n) : p_k, \mathsf{Enc}_{p_k}(m)\}$
  - $\{p_k, s_k \leftarrow \mathsf{Gen}(1^n) : p_k, \mathcal{S}(p_k, 1^n)\}$
- $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is secure *if and only if* it is Zero-Knowledge.

Zero-Knowledge interactions

Suppose Alice (the prover) would like to convince Bob (the verifier) that a particular string x is in a language L. Since Alice does not trust Bob, Alice wants to perform this proof in such a way that Bob learns nothing else except that $x \in L$. In particular, it should not be possible for Bob to later prove that $x \in L$ to someone else.

Examples:

- I know p and q, the prime factors of N.
- I am of drinking age.
- The two balls you are holding (blindfolded) are of different colours.

Interactive protocols

- *Interactive Turing Machine:* read-only *input*, read-only *auxiliary input*, read-only *random source*, read-only *receiving channel*, write-only *sending channel* and finally an output.
- A protocol (A, B) is a pair of ITMs with common input (as of now) sharing communication channels.
- Let $M_A = \{m_A^1, m_A^2, \ldots\}$, $M_B = \{m_B^1, m_B^2, \ldots\}$, and let $x, r_1, r_2, z_1, z_2 \in \{0,1\}^*$. The pair $((x, r_1, z_1, M_A), (x, r_2, z_2, M_B))$ is an *execution protocol* if on common input x, with auxiliary input z_i and random input r_i respectively, results in m_A^i being the i^{th} message received by A and m_B^i being the i^{th} message received by B. We denote this (execution/view) by $A(x, z_1) \leftrightarrow B(x, z_2)$ (or sometimes simply as (A, B)).
- (M_A, M_B) is the *transcript* of the execution.
- $\text{out}_X((A, B)), X \in \{A, B\}$ is the output of A or B.

Interactive proofs

A pair of interactive machines (P, V) is an interactive proof system for a language L if V is a p.p.t. machine and the following properties hold.

- (**Completeness**) For every $x \in L$, there exists a witness string $y \in \{0,1\}^*$ such that for every auxiliary string z:

$$\Pr\left[\mathrm{out}_V\left[P(x,y) \leftrightarrow V(x,z)\right] = 1\right] = 1$$

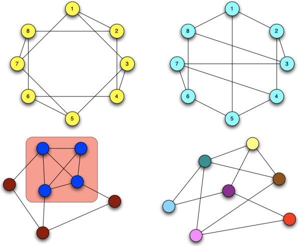
- (**Soundness**) There exists some negligible function ϵ such that for all $x \notin L$ and for all prover algorithms P^*, and all auxiliary strings $z \in \{0,1\}^*$,

$$\Pr\left[\mathrm{out}_V\left[P^*(x) \leftrightarrow V(x,z)\right] = 0\right] = 1 - \epsilon(|x|)$$

(We may replace $1 - \epsilon(|x|)$ by some constant $1/2$.)

Interactive proofs and computational complexity

- It trivially holds that $\mathbf{NP} \subset \mathbf{IP}$.
- Surprisingly, there are languages that are not known to be in **NP** that also have interactive proofs. *Graph non-isomorphism is an example*. Isomorphic if $\exists \sigma$ such that $\sigma(G_1) = G_2$.



- $\mathbf{IP} = \mathbf{PSPACE}$. (Shamir)

Interactive proof for graph non-isomorphism

PROTOCOL 118.3: PROTOCOL FOR GRAPH NON-ISOMORPHISM

Input:	$x = (G_0, G_1)$ where $\lvert G_i \rvert = n$
$V \xrightarrow{H} P$	The verifier, $V(x)$, chooses a random bit $b \in \{0,1\}$, chooses a random permutation $\sigma \in S_n$, computes $H \leftarrow \sigma(G_b)$, and finally sends H to the prover.
$V \xleftarrow{b'} P$	The prover computes a b' such that H and $G_{b'}$ are isomorphic and sends b' to the verifier.
$V(x, H, b, b')$	The verifier accepts and outputs 1 if $b' = b$ and 0 otherwise. Repeat the procedure $\lvert G_1 \rvert$ times.

Completeness is obvious. Soundness follows from the fact that a cheating prover succeeds by probability at most 2^{-n}.

Efficient provers

- An interactive proof system (P, V) is said to have an *efficient prover* with respect to the witness relation R_L if P is p.p.t. and the completeness condition holds for every $y \in R_L(x)$.
- The soundness condition still requires that not even an all powerful prover strategy P^* can cheat the verifier V. A more relaxed notion – called an interactive argument considers only P^*'s that are n.u. p.p.t.

An interactive protocol for graph isomorphism

PROTOCOL 120.6: PROTOCOL FOR GRAPH ISOMORPHISM

Input:	$x = (G_0, G_1)$ where $\|G_i\| = n$
P's **witness**:	σ such that $\sigma(G_0) = G_1$
$V \xleftarrow{H} P$	The prover chooses a random permutation π, computes $H \leftarrow \pi(G_0)$ and sends H.
$V \xrightarrow{b} P$	The verifier picks a random bit b and sends it.
$V \xleftarrow{\gamma} P$	If $b = 0$, the prover sends π. Otherwise, the prover sends $\gamma = \pi \cdot \sigma^{-1}$.
V	The verifier outputs 1 if and only if $\gamma(G_b) = H$.
P, V	Repeat the procedure $\|G_1\|$ times.

The protocol is also *Zero-Knowledge*. V does not learn about σ.

Honest verifier Zero-Knowledge

- Let (P, V) be an efficient interactive proof for the language $L \in \mathbf{NP}$ with witness relation R_L. (P, V) is said to be *Honest Verifier Zero-Knowledge* if there exists a p.p.t. simulator $\mathcal{S}$ such that for every n.u. p.p.t. distinguisher $\mathcal{D}$, there exists a negligible function $\epsilon()$ such that for every $x \in L$, $y \in R_L(x)$, $z \in \{0,1\}^*$, $\mathcal{D}$ distinguishes the following distributions with probability at most $\epsilon(n)$.

  - $\{\text{view}_V [P(x,y) \leftrightarrow V(x,z)]\}$
  - $\{S(x,z)\}$

- Intuitively, the definition says whatever V "saw" in the interactive proof could have been generated by V himself by simply running the algorithm $\mathcal{S}(x,z)$.

Zero-Knowledge

- Let (P, V) be an efficient interactive proof for the language $L \in \mathbf{NP}$ with witness relation R_L. (P, V) is said to be *Zero-Knowledge* if for every p.p.t adversary V^*, there exists an expected p.p.t. simulator $\mathcal{S}$ such that for every n.u. p.p.t. distinguisher $\mathcal{D}$, there exists a negligible function $\epsilon()$ such that for every $x \in L$, $y \in R_L(x)$, $z \in \{0,1\}^*$, $\mathcal{D}$ distinguishes the following distributions with probability at most $\epsilon(n)$.

  - $\{\text{view}_{V^*}[P(x,y) \leftrightarrow V^*(x,z)]\}$
  - $\{S(x,z)\}$

- *Perfect Zero-Knowledge* if the two distributions are identical.
- An alternate formalization more directly considers what V^* "can do", instead of what V^* "sees". Just change view_{V^*} to out_{V^*}. However, completely equivalent.

The interactive protocol for graph isomorphism is *perfect zero knowledge*

123.4: SIMULATOR FOR GRAPH ISOMORPHISM

1. Randomly pick $b' \leftarrow \{0,1\}$, $\pi \leftarrow S_n$
2. Compute $H \leftarrow \pi(G_{b'})$.
3. Emulate the execution of $V^*(x,z)$ by feeding it H and truly random bits as its random coins; let b denote the response of V^*.
4. If $b = b'$ then output the view of V^*—i.e., the messages H, π, and the random coins it was feed. Otherwise, restart the emulation of V^* and repeat the procedure.

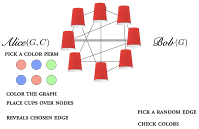
- the expected running time of $\mathcal{S}$ is polynomial.
- In the execution of $\mathcal{S}(x,z)$, H is identically distributed to $\pi(G_0)$, and $\Pr[b = b'] = 1/2$.

Every language in NP has a zero-knowledge proof

- **Step 1** Show a ZK proof (P', V') (with efficient provers) for an **NP**-Complete language (say *Graph 3-colouring*).
- **Step 2** For a given language $L \in \mathbf{NP}$, an instance x and a witness y:
  1. Both P and V use Cook's reduction to x to an instance x' of *Graph 3-colouring*. They get the same x' since the reduction is deterministic.
  2. Ditto with y to obtain a witness y' for the instance x'.
  3. Use Step 1.

ZKP of *Graph 3-colouring*

Given: A graph (V, E) and a colouring C of the vertices.



1. The prover picks a random permutation π over the colours $\{1, 2, 3\}$.
2. The prover colours the vertices with the permuted colours and covers the colours.
3. The verifier is then asked to pick a random edge, the prover uncovers the connected vertices and demonstrates that they are differently coloured.
4. If the procedure (the 3 steps above) is repeated $O(n|E|)$ times then the soundness error will be 2^{-n}.

Commitments

Commit: Put a value v in a locked box and give away the box.

Reveal: At a later time unlock an reveal v.

- A polynomial-time machine Com is called a *commitment scheme* it there exists some polynomial $l()$ such that the following two properties hold:
  1. **Binding:** For all $n \in \mathbb{N}$ and all $v_0, v_1 \in \{0,1\}^n$, $r_0, r_1 \in \{0,1\}^{l(n)}$, it holds that $\text{Com}(v_0, r_0) \neq \text{Com}(v_1, r_1)$.
  2. **Hiding:** For every n.u. p.p.t. distinguisher $\mathcal{D}$, there exists a negligible function $\epsilon()$ such that for every $n \in \mathbb{N}$, $v_0, v_1 \in \{0,1\}^n$, $\mathcal{D}$ distinguishes the following distributions with probability at most $\epsilon(n)$.
     - $\{r \leftarrow \{0,1\}^{l(n)} : \text{Com}(v_0, r)\}$
     - $\{r \leftarrow \{0,1\}^{l(n)} : \text{Com}(v_1, r)\}$
- If one-way permutations exist, then commitment schemes exist.

Pedersen commitment

Setup:

1. Receiver chooses two large primes p (typically 1024 bits) and q (typically 160 bits) such that $q|p-1$. Receiver also chooses g which has order q
2. Receiver chooses a secret $a \in \mathbb{Z}_q$. Let $h = g^a \bmod p$
3. $\langle p, q, g, h \rangle$ are public parameters. a is a secret parameter
4. We have $g^q = 1 \bmod p$. Also, $\langle g \rangle = \{g, g^2, g^3, \ldots, g^q = 1\}$

Commit: To commit $x \in \mathbb{Z}_q$, sender chooses $r \in \mathbb{Z}_q$, and sends $c = g^x h^r \bmod p$

Open: To open sender reveals x and r, receiver verifies $c \stackrel{?}{=} g^x h^r \bmod p$

Pedersen commitment

- Unconditionally hiding
  1. Given c, every x is equally likely
  2. Given x, r and any x', there exist r' such that $g^x h^r = g^{x'} h^{r'}$. In fact $r' = (x - x')a^{-1} + r \bmod q$
- Computationally binding
  1. Suppose sender cheats by opening another $x' \neq x$. That is sender finds r' such that $g^x h^r = g^{x'} h^{r'}$.
  2. Then sender can compute $\log_g h = (x - x') \cdot (r - r')^{-1}$. Assuming Discrete Log is hard, this is computationally hard for the sender.

ZKP of Pedersen commitment

- Public commitment $c = g^x h^r \mod p$
- Private knowledge x, r
- Protocol:
  1. P picks random $y, s \in \mathbb{Z}_q$, sends $d = g^y h^s \mod p$
  2. V sends a random challenge $e \in \mathbb{Z}_q$
  3. P sends $u = y + ex,\ v = s + er \mod q$
  4. V accepts if $g^u h^v = dc^e \mod p$
- Soundness and completeness?

Applications of Zero Knowledge: Proof of Knowledge

- Login to a server with a password.
- Login to a server with a secret key:
  - User sends "login id"
  - Server sends $\sigma = (\text{"Server name"}, r)$.
  - User signs σ with secret key.
  - Server verifies with user's public key.
- User simply proves in Zero-Knowledge that it knows the key S corresponding to V.