Foundations of Probabilistic Proofs (Fall 2022)
Note 5: Zero-Knowledge IPs
Date: 2022.10.04

This note contains definitions, theorems, facts, *etc.* that are not fully explained in lectures due to limited time. If you think there are anything missing or any mistakes, please contact ziyi.guan@epfl.ch.

1 Rewinding power of simulators

We define malicious-verifier zero-knowledge (MVZK) in lecture and present a MVZK interactive proof system for Graph Isomorphism. The protocol is as follows:

Prover $P((G_0, G_1), \sigma)$	**Verifier** $V(G_0, G_1)$
Sample random permutation $\phi : [n] \to [n]$	
Send $H \coloneqq \phi(G_0)$	
	Sample $b \leftarrow^{\$} \{0, 1\}$
Send $\psi \coloneqq \phi \circ \sigma^b$	
	Check if $H = \psi(G_b)$

The simulator $S(\tilde{V}, (G_0, G_1))$ for this protocol is defined as:

1. Sample a random bit $b \in \{0, 1\}$,
2. Sample random permutation $\psi : [n] \to [n]$,
3. Compute $H \coloneqq \psi(G_b)$,
4. Give H to $\tilde{V}$ and get $\tilde{b}$,
5. If $\tilde{b} \neq b$ go to step 1,
6. Output $((G_0, G_1), H, \tilde{b}, \psi)$.

In step 5, S repeats the whole process if the bit sampled and the bit received by the $\tilde{V}$ are different. In particular, S works since it uses rejection sampling. i.e. it rejects and rewinds until the condition $\tilde{b} = b$ is met. By rejection sampling, we mean that we throw away "bad" samples during the sampling process. For example, say we have a square and an inscribed circle, and we want to sample points from the circle. To do that, we can sample points from the square, and throw away the samples outside of the circle. Using this technique, the probability distribution of the view of the simulator and the malicious verifier is the same when $\tilde{b} = b$.

We note that almost all simulators for malicious verifiers use the rewinding strategy. Now we give a general template of how rewinding simulators work.

- When proving zero-knowledge of an interactive protocol, we consider a malicious verifier $\tilde{\mathcal{V}}$ and an *honest prover* $\mathcal{P}$.
- $\mathcal{P}$ knows some information that $\tilde{\mathcal{V}}$ does not, and we want to hide this information during the protocol from the (cheating) verifier $\tilde{\mathcal{V}}$.
  - Formally, we call it the zero-knowledge property: whatever $\tilde{\mathcal{V}}$ outputs from this interaction, $\tilde{\mathcal{V}}$ could have generated without interacting with $\mathcal{P}$ at all.

  - This prevents $\tilde{\mathcal{V}}$ from learning any new information through interactions.

- To prove the zero-knowledge property, we need to show that given the distribution of the interaction transcript between an honest prover and the malicious verifier $\tilde{\mathcal{V}}$, we can sample the same (or close) distribution without interactions with any honest provers. Moreover, we only have black-box access to $\tilde{\mathcal{V}}$, i.e. we will ask as the honest prover and send messages to $\tilde{\mathcal{V}}$.

- Rewinding means that, during the process of sending simulated messages $z_1, z_2, \ldots, z_m$ to $\tilde{\mathcal{V}}$, it is possible to go back to a previous state of $\tilde{\mathcal{V}}$ and send in a different z_i for all $i \in [m]$.

2 Complexity classes and relationships

In the lecture, we mention some limitations of zero-knowledge interactive proofs. In particular, we talk about how adding the extra property of zero-knowledge will significantly decrease the power of IP. We present the formal definitions of the complexity classes mention and their relationships with previously introduced complexity classes, and IP with zero-knowledge.

First, we recap the limitations we discuss in lecture:

- HVZK-IP: all languages in IP that has honest-verifier zero-knowledge.

- MVZK-IP: all languages in IP that has malicious-verifier zero-knowledge.

- MVZK-IP $\subseteq$ HVZK-IP $\subseteq$ IP: clear from definitions.

- BPP $\subseteq$ MVZK-IP.

  - Similar to P, but defined using polynomial-time *probabilistic Turing machines* (PTM).
  - A probabilistic Turing machine differ from a plain Turing machine in that it has multiple transition functions, and it chooses between them according to some probabilistic distribution.
    * A PTM can have stochastic results in terms of both runtime and acceptance of a particular input.
    * The expected runtime of a PTM refers to the average runtime of this machine.
    * The runtime of a PTM refers to the worst case runtime, i.e. the upper bound of the runtime in all executions of the machine.
    * There will be acceptance probability associated with a PTM. Also, a PTM may behave different with respect to the same input.
  - GI $\in$ MVZK-IP as shown in lecture, but GI is not known to be in BPP: a possible gap.
  - We have seen one problem in BPP before: Polynomial identity testing (PIT), which is resolved by applying the Schwartz-Zippel Lemma [1].

- MVZK-IP $\subseteq$ HVZK-IP $\subseteq$ AM $\cap$ coAM.

[1] The Schwartz-Zippel lemma is defined in note 2

– Arthur-Merlin (AM): The class of decision problems for which a "yes" answer can be verified by an Arthur-Merlin protocol, as follows:

Merlin	**Arthur(x)**
	Generate a challenge based on x
	Generate randomness
	Send the challenge and the randomness
Send a response based on the messages	
	Decides whether to accept

where Arthur is a BPP verifier with an algorithm such that

  * If the answer if "yes", then Merlin can act in such a way that Arthur accepts with probability at least 2/3.
  * If the answer is "no", then however Merlin acts, Arthur will reject with probability at least 2/3.

– AM ∩ coAM: The class of decision problems for which both "yes" and "no" answers can be verified by an AM protocol.

  * AM ∩ coAM is believed to be small compared to PSPACE: Having zero-knowledge property reduces the power of IP.

3 Computational indistinguishability

In the lecture, we show the limitations of zero-knowledge IP. One option to overcome the limitations is to relax the requirement on the view sampled by the simulator. More specifically, instead of requiring S to sample a view that has the same distribution as the actual view, we require $S(\tilde{V}, X)$ to be computationally close to the actual view $\text{view}(\langle P, \tilde{V}\rangle(X))$. We now formalise this notion of computational closeness, or, more commonly referred to, the computational indistinguishability.

Definition 1. *Let $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ be two family of distributions in which n is the length of input. We say that they are* **computationally indistinguishable** *if for any probabilistic polynomial-time algorithm A such that*

$$\delta(n) = \left| \Pr_{x\leftarrow X_n}[A(x) = 1] - \Pr_{x\leftarrow Y_n}[A(x) = 1] \right|,$$

where $\delta(n)$ is a negligible function in n.

Remark 1. *We do not have a formal definition for negligible functions. In general, we say a function f is negligible if for every positive polynomial $\mathsf{poly}(\cdot)$, there exists a positive integer N_{poly} such that for all $x > N_{\mathsf{poly}}$,*

$$|f(x)| < \frac{1}{\mathsf{poly}(x)}.$$

Example 1. *Consider family of distributions $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ such that*

$$X_n = \begin{cases} 0^n & \text{with probability } 2^{-n}, \\ 1^n & \text{with probability } 1 - 2^{-n}; \end{cases} \quad \text{and } Y_n = 1^n.$$

We claim that $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are* **computationally indistinguishable**.

Proof. The main idea is that X_n would be 1^n most of the times, in other words negligible for a reasonable n. Since Y_n is 1^n all the time, the probability difference is negligible. □