Interactive Proofs and Zero Knowledge

Notes by Yael Kalai (with edits by Henry Corrigan-Gibbs)

MIT - 6.5610
Lecture 10 (March 3, 2025)

Warning: This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Interactive proofs
- Zero-knowledge interactive proofs
- Commitment schemes from LWE

Recap

In the last two classes we learned about fully homomorphic encryption (FHE) schemes. FHE is an important primitive in today's world where we have large amounts of (possibly sensitive) data that we cannot store locally on our own devices. FHE allows us to store our data encrypted on an untrusted platform, and yet allow the platform to do computations on our encrypted data. Moreover, even if our sensitive data is stored on a trusted platform, for example our medical data is stored on our hospital servers, which we supposedly trust, the hospitals may want to collaborate and learn from their joint sensitive medical data. FHE allows them to collaborate without revealing to each other sensitive information about each other's data. FHE indeed allows us to obtain secrecy but what about integrity?

Suppose we store our data on an untrusted platform and then request the platform to perform computations on our encrypted data. How do we know that indeed the platform is doing the instructed computation? In other words, *can we efficiently verify that a computation was done correctly?* Namely, is there a *succinct* and *efficiently verifiable* proof that we can append to the output of a computation attesting to the fact hat this output is indeed correct? This is the topic for the next five lectures.

Following our convention that "efficient" means polynomial time, we ask which computations (beyond P) have proofs of correctness that can be verified in polynomial time? This is precisely the definition of the complexity class NP, which is the set of all languages that have membership proofs (aka witnesses) that can be checked by a polynomial-time verifier algorithm, denoted $\mathcal{V}$. We would like proofs of correctness for languages outside of NP. More precisely, our focus is on a fine-grained version of the question above. Namely, *do there exist proofs of correctness for time T computations that can be verified in time $<< T$* (say time T^{ϵ} or even polylog(T))? Unfortunately, we do not believe that every T-computable language has a proof (or a "witness") of size $<< T$. As you learned by now, cryptography is an art of overcoming such barriers. We overcome this barrier by changing the definition of a proof and by making use of some cryptographic magic!

Interactive Proofs and Zero-Knowledge Proofs

Proof systems have been studied by mathematicians for thousands of years, starting from Euclid (300 BCE). Yet, until recently, all proof systems were of a somewhat similar form which is simply a list of formulas that follow from a set of inference rules and axioms. This changed in the mid eighties when Goldwasser, Micali and Rackoff defined the notion of a *zero-knowledge proof* [2]. Intuitively, a zero-knowledge proof is one that reveals no information beyond the validity of the statement (and can be verified in polynomial time). What does "no information" mean? How is this formalized? Goldwasser et al. formalized it as follows: No information means we could have generated it on our own. However, with such formulation zero-knowledge proofs exist only for easy languages (i.e., ones that are in P).

To avoid this limitation, they completely changed the way we think about proofs. They defined a new notion called *interactive proofs*. Such proofs extend upon the classical notion of "proofs" in two ways. First, rather than solely considering a verifier algorithm $\mathcal{V}$, we instead think of the proof as arising from *interaction* between the verifier $\mathcal{V}$ and a prover algorithm $\mathcal{P}$. Second, we allow the verifier to access "private" randomness that is not accessible to the prover.

Both the verifier and prover algorithms will have access to the input of the problem instance. The two algorithms will exchange messages sequentially, computing the next message in the sequence as a function of the messages up to that point. Ultimately, the verifier algorithm will decide whether to accept or reject the problem instance. We can think of the interaction metaphorically as the prover trying to "convince" the verifier of the problem instance being true, and of the verifier trying to verify that the prover is not "dishonest" or "cheating" and misleading the verifier into accepting a false statement.

Definition 1 (Interactive Proof system (IP))**.** An interactive proof system for a language L consists of an interactive p.p.t. verifier algorithm $\mathcal{V}$ and an interactive (possibly inefficient) algorithm $\mathcal{P}$, which exchange a series of *messages* $m_1, \ldots, m_k$, with each message computed as a function of all the previous messages: $m_i = \mathcal{V}(x, m_1, \ldots, m_{i-1})$, and likewise for $\mathcal{P}$. Notably, the verifier's computations may also depend upon private random bits not revealed to the prover. Denote by $(\mathcal{P}, \mathcal{V}(r))(x) = 1$ the event that the verifier $\mathcal{V}$, with private randomness r, accepts the interactive proof after communicating with the prover $\mathcal{P}$ on the joint input x and assuming $\mathcal{V}$ has randomness r. The following two properties are required to hold:

1. *Completeness*: $\forall x \in L$,

$$\Pr[(P, V(r))(x) = 1] \geq \frac{2}{3}$$

2. *Soundness*: $\forall x \notin L$ and $\forall$ (malicious and possibly all powerful) $\mathcal{P}^*$,

$$\Pr[(P^*, V(r))(x) = 1] \leq \frac{1}{3}.$$

Remark. These numbers ($\frac{2}{3}$ and $\frac{1}{3}$) are arbitrary. By repeating the interactive proof λ times and accepting if and only if at least $\frac{\lambda}{2}$ are accepting we can get completeness $1 - \text{negl}(\lambda)$ and soundness $\text{negl}(\lambda)$. This follows from the Chernoff bound, which is a concentration bound that says that if $X_1, \ldots, X_\lambda$ are independent and identically distributed Bernouli random variables such that $\Pr[X_i = 1] = p$ then $\Pr[|\frac{1}{\lambda} \sum_{i \in \lambda} X_i - p| > \delta] \leq 2^{-O(\delta^2 \cdot p \cdot \lambda)}$.

See this for information about the Chernoff bound.

The class IP is the set of all languages L that have such an interactive proof. Note that $\mathsf{NP} \subseteq \mathsf{IP}$ but IP may contain additional languages.

The Importance of both interaction and randomness

We may ask what happens when either of these two properties (randomization or interaction) are removed. If we remove randomization, then the resulting class is NP. This is the case since then the algorithms are deterministic, and hence the transcript of any interaction between any prover-verifier pair $(\mathcal{P}, \mathcal{V})$ could be generated by a prover $\mathcal{P}'$ and given to a verifier $\mathcal{V}'$. The unknown randomness from the verifier is key to the additional power of IP, as it is something that an all-powerful prover cannot generate on its own. In contrast, we can, without loss of generality, take the prover to be deterministic.

If, instead, we only remove interaction, then we get a complexity class called Merlin-Arthur, or AM. It is not known whether $\mathsf{AM} = \mathsf{NP}$ or whether $\mathsf{AM} = \mathsf{IP}$, but we believe that $\mathsf{AM} = \mathsf{NP}$.

Is IP *more expressive than* NP*?*

It turns out that interactive proofs are actually very powerful:

Theorem 2 (Shamir '90 [3])**.** IP = PSPACE.

Let's go back to our story about zero-knowledge proofs. It was shown by Goldreich, Micali and Wigderson [1] that every proof can be converted into a zero-knowledge interactive proof. Namely, they presented a zero-knowledge proof for every language in NP. Moreover, the prover is efficient if it is given a valid witness as input. These proofs are widely used in cryptography. For example, they are used for authentication, where a user wants to prove that he knows a secret key corresponding to a given public key. We need this proof to be zero-knowledge. Actually, one of the most widely used digital signature scheme (ECDSA) uses the idea of zero-knowledge proofs. We will not focus on constructions of zero-knowledge proofs since it is covered in other cryptography classes at MIT (such as 6.5620 and 6.1600), but let me give you a high-level idea since it is too simple and beautiful to omit entirely.

Zero-knowledge proofs for NP

What does it mean to "learn nothing" from an interaction with someone? Cryptography gives a simple, beautiful, and useful definition. The basic idea is that you have learned nothing from a conversation if you could have sat at home by yourself and written down a transcript of the conversation (without speaking with anyone else).

The cryptographic formalization of zero knowledge for an interactive proof system looks like this:

Definition 3 (Zero knowledge)**.** *Zero knowledge*: $\forall$ (malicious and possibly all powerful) $\mathcal{V}^*$, there exists a p.p.t algorithm Sim ("the simulator") such that $\forall x \in L$,

$$\{\mathsf{View}_{\mathcal{V}*}(\mathcal{P}(x), \mathcal{V}^*)\} \approx \{\mathsf{Sim}(x)\},$$

where $\mathsf{View}_{\mathcal{V}*}(\mathcal{P}(x), \mathcal{V}^*)$ denotes the "view" of the malicious verifier $\mathcal{V}^*$ in its interaction with the prover—this is the verifier's randomness and all of the message it receives from the prover $\mathcal{P}$ running on input x.

We will see a zero-knowledge proof for a specific NP-complete language called 3Col which contains the set of all graphs $G = (V, E)$ such that the set of vertices V can be colored by three colors: $C : V \to \{1, 2, 3\}$ such that no two adjacent vertices have the same color. Namely, for every $(u, v) \in E$, $C(u) \neq C(v)$.

We convert a coloring $C : V \to \{1,2,3\}$, which is a proof that the graph G is 3-colorable, into a zero-knowledge proof, as follows:

1. The prover does the following:
   (a) Choose a random permutation $\pi : \{1,2,3\} \to \{1,2,3\}$. Denote by $V = \{1,2,\ldots,n\}$.
   (b) For every $i \in [n]$ place the color $\pi(C(i))$ in an opaque locked box and send the n locked boxes to the verifier.
2. The verifier chooses a random edge $(i,j) \in E$ and sends (i,j) to the prover.
3. The prover sends the keys that open only box i and box j.
4. The verifier accepts if and only if the colors in these boxes are distinct and are legal (i.e., belong to the set $\{1,2,3\}$).

Is this zero-knowledge? Yes, the only thing the verifier learned is two distinct random colors. The verifier could have simulated the prover's last message on its own. It has completeness 1. The soundness is only $1 - \frac{1}{|E|}$ but can be amplified via repetitions. Each time we repeat we need to choose a fresh permutation.

Remark. The above protocol is a physical protocol where the prover sends opaque locked boxes. Such boxes have a digital analogue. This is called a commitment scheme.

Commitment Schemes

Definition 4. A commitment scheme corresponding to a message space $\mathcal{M}$ consists of a pair of algorithms $(\mathsf{Gen}, \mathsf{Com})$:

- Gen is a p.p.t. algorithm that takes as input the security parameter 1^λ and outputs public parameters, denoted by $\mathsf{pp} \in \{0,1\}^N$ where $N = \text{poly}(\lambda)$.
- Com is a polynomial-time computable function that takes an input public parameters pp, a message $m \in \mathcal{M}$ and randomness $r \overset{R}{\leftarrow} \{0,1\}^\lambda$ and outputs a commitment $\mathsf{Com}(\mathsf{pp}, m, r)$.

The following two properties are required to hold:

- **Hiding:** For every $m_0, m_1 \in \mathcal{M}$,
$$(\mathsf{pp}, \mathsf{Com}(\mathsf{pp}, m_0, r_0)) \approx (\mathsf{pp}, \mathsf{Com}(\mathsf{pp}, m_1, r_1))$$
for $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$ and $r_0, r_1 \overset{R}{\leftarrow} \{0,1\}^\lambda$.
- **Binding:** For every p.p.t. adversary $\mathcal{A}$:
$$\Pr_{\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)}[A(\mathsf{pp}) = (m_0, r_0, m_1, r_1) \text{ s.t. } m_0 \neq m_1 \ \wedge \ \mathsf{Com}(\mathsf{pp}, m_0, r_0) = \mathsf{Com}(\mathsf{pp}, m_0, r_0) = \text{negl}(\lambda).$$

One can think of both Gen and Com as randomized algorithms. We chose to explicitly include the randomness of Com, and hence think of it as being deterministic since when the commitment is opened the randomness is revealed

Construction from the LWE *assumption*

In what follows we construct a commitment scheme for the message space $\mathcal{M} = \{0,1\}$.

- $\mathsf{Gen}(1^\lambda)$ chooses a random matrix $A \xleftarrow{R} \mathbb{Z}_q^{m+n}$ and a random vector $u \xleftarrow{R} \mathbb{Z}_q^m$. It outputs $\mathsf{pp} = (A,u)$
- $\mathsf{Com}((A,u), b, (s,e)) = As + e + bu$

Note that the hiding property follows directly from the LWE assumption. The binding property is statistical. Namely,

$$\Pr_{A,u}[\exists s_0, s_1 \in \mathbb{Z}^n,\ e_0, e_1 \in [-B,B]^m \ : \ As_0 + e_0 = As_1 + e_1 + u] = \mathrm{negl}(\lambda).$$

This follows from a simple counting argument assuming $n \log q + m \log B << m \log q$.

Remark. We know how to construct a commitment scheme from any one-way function (which is a minimal assumption) but the analysis is quite complicated. The resulting commitment scheme is also statistically binding and computationally hiding. We also know how to construct commitment schemes that are statistically hiding and computationally binding from any one-way function. These schemes are interactive and complicated.

References

[1] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1986.

[2] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[3] Adi Shamir. Ip=pspace. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 11–15. IEEE Computer Society, 1990.