Lecture 17: Zero-Knowledge Interactive Proofs

Notes by Yael Kalai

MIT - 6.5620

Lecture 17 (November 5, 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

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). A huge amount of progress in computer science has been made by attempting to computationally formalize the notion of a proof. This led to the definition of the complexity class NP, which is the set of all languages L such that membership in the language $x \in L$ can be efficiently verified given a polynomial-size proof $w \in \{0,1\}^{\text{poly}(n)}$ (called a witness).

Definition 1. A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $m : \mathbb{N} \to \mathbb{N}$ and a poly-time algorithm R such that for every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$, $x \in L$ if and only if there exists $w \in \{0,1\}^{m(n)}$ such that $R(x,w) = 1$.

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* [1]. 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 2 (Interactive Proof system (IP))**.** An interactive proof system for a language L consists of an interactive PPT 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*: $\forall x \in L$,
$$\Pr[(\mathcal{P}, \mathcal{V}(r))(x) = 1] \geq \frac{2}{3}$$
2. *Soundness*: $\forall x \notin L$ and $\forall$ (malicious and possibly all powerful) $\mathcal{P}^*$,
$$\Pr[(\mathcal{P}^*, \mathcal{V}(r))(x) = 1] \leq \frac{1}{3}.$$

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.

We therefore often give $\mathcal{P}$ and $\mathcal{V}$ an additional input 1^λ, and require completeness to hold with probability $1 - \text{negl}(\lambda)$ and soundness to hold with probability $\text{negl}(\lambda)$.

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

Remark. The power of the class IP comes from the fact that the verifier is randomized. If the verifier was deterministic this class would be equivalent to NP

Example: Graph Non-Isomorphism

A graph G is defined by a set of vertices V and a set of edges $E \subseteq V \times V$, where each edge is a pair (u, v) for $u, v \in V$. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be isomorphic if there exists a permutation $\pi : V_1 \to V_2$ such that for every $u, v \in V_1$,

$$(u, v) \in E_1 \quad \text{if and only if} \quad (\pi(u), \pi(v)) \in E_2.$$

Formally,we have defined a directed graph,because we have denoted an edge as an ordered pair. For this lecture this distinction is not important and you can think of the edges as non-directed.

Denote by

$$L_{\mathsf{iso}} = \{(G_1, G_2) : \ G_1, G_2 \ \text{ are isomorphic}\}.$$

Note that $L_{\mathsf{iso}} \in \mathsf{NP}$. But we believe that the complement language:

$$\bar{L}_{\mathsf{iso}} = \{(G_1, G_2) : \ G_1, G_2 \ \text{ are not isomorphic}\}$$

is not in NP.

Yet there is a simple interactive proof for proving membership in $\bar{L}_{\mathsf{iso}}$. Given a pair of non-isomorphic graphs (G_1, G_2) the interactive proof proceeds as follows: In what follows, assume for simplicity that $V_1 = V_2 = [n]$. This is without loss of generality since we can simply rename the variables to be in $[n]$ assuming $|V_1| = |V_2| = n$.[1]

[1] If $|V_1| \neq |V_2|$ then the verifier immediately knows that the two graphs are not isomorphic.

1. V chooses a random $b \in \{1, 2\}$ and chooses a random permutation $\pi : [n] \to [n]$. It sends $\pi(G_b)$ to P.
2. $\mathcal{P}$ sends b' such that $G_{b'}$ is isomorphic to $\pi(G_b)$.
3. The verifier accepts if and only if $b' = b$.

This shows that interactive proofs are likely to be more powerful than NP! There is something else really nice about this proof: the (honest) verifier did not learn anything about why the graphs are non-isomorphic! He knew that $b' = b$!

Definition of Zero-Knowledge

Definition 3 (Honest verifier ZK)**.** Fix a language $L \in \mathsf{NP}$ with a corresponding NP relation R. We say that an interactive proof $(\mathcal{P}, \mathcal{V})$ for L is *honest-verifier (computational) zero-knowledge* if there exists a PPT simulator $\mathcal{S}$ such that for every polynomials $\ell_1 = \ell_1(\lambda)$ and $\ell_2 = \ell_2(\lambda)$, the following holds:

For every $\lambda \in \mathbb{N}$ every $x \in L$ of size $|x| = \ell_1(\lambda)$, every witness w with $(x, w) \in R$, and every auxiliary input $z \in \{0,1\}^{\ell_2(\lambda)}$, the following ensembles (over the security parameter) are computationally indistinguishable:

$$\left\{ \mathsf{View}\Big((\mathcal{P}(w), \mathcal{V})(1^\lambda, x, z) \Big) \right\}_{\lambda \in \mathbb{N}} \approx \left\{ S(1^\lambda, x, z) \right\}_{\lambda \in \mathbb{N}}.$$

In addition, it is required that $\mathcal{P}$ runs in polynomial time (given a witness w).

Definition 4 (ZK)**.** Fix a language $L \in \mathsf{NP}$ with a corresponding NP relation R. We say that an interactive proof $(\mathcal{P}, \mathcal{V})$ for L is *(computational) zero-knowledge* if for every PPT verifier $\mathcal{V}^*$ there exists a PPT simulator $\mathcal{S}$ such that for every polynomials $\ell_1 = \ell_1(\lambda)$ and $\ell_2 = \ell_2(\lambda)$, the following holds:

For every $\lambda \in \mathbb{N}$ every $x \in L$ of size $|x| = \ell_1(\lambda)$, every witness w with $(x, w) \in R$, and every auxiliary input $z \in \{0,1\}^{\ell_2(\lambda)}$, the following ensembles (over the security parameter) are computationally indistinguishable:

$$\left\{ \mathsf{View}\Big((\mathcal{P}(w), \mathcal{V}^*)(1^\lambda, x, z) \Big) \right\}_{\lambda \in \mathbb{N}} \approx \left\{ S(1^\lambda, x, z) \right\}_{\lambda \in \mathbb{N}}.$$

In addition, it is required that $\mathcal{P}$ runs in polynomial time (given a witness w).

Construction of ZK Protocol for the NP *Language* QR

Goldwasser Micali and Rackoff [1] showed how to construct ZK proof for some NP languages. For example, they constructed a ZK proof for the language

$$\mathbb{QR} = \{(N, y) : \exists x \in \mathbb{Z}_N^* \text{ s.t. } y = x^2 \bmod N\}$$

The proof is very simple! Given an instance (N, y) it proceeds as follows:

1. The prover chooses a random $r \leftarrow \mathbb{Z}_N^*$ and sends to the verifier the element $s = r^2 \bmod N$.
2. The verifier chooses a random bit $b \leftarrow \{0,1\}$.
3. If $b = 0$ the prover sends $z = r$. If $b = 1$ the prover sends $z = r \cdot x$, where x is the witness (i.e., $y = x^2 \bmod N$).
4. If $b = 0$ the verifier accepts iff $z^2 = s \bmod N$. If $b = 0$ the verifier accepts iff $z^2 = s \cdot y \bmod N$.

This protocol is ZK. Let us first prove that it is honest-verifier ZK. The simulator will choose a random bit $b \leftarrow \{0,1\}$ on behalf of the verifier. If $b = 0$ it will first choose the prover's first message, by sampling a random $r \leftarrow \mathbb{Z}_N^*$ and setting $s = r^2 \bmod N$, and then set the prover's second message to be $z = r$. If $b = 1$ then it will first choose the prover's second message, by sampling a random element $z \leftarrow \mathbb{Z}_N^*$, and then set the prover's first message to be $s = z^2 \cdot y^{-1} \bmod N$.

This protocol is also (malicious verifier) ZK. The simulator is very similar to the one above. It will guess the (malicious) verifier's bit $b \leftarrow \{0,1\}$, and produce (s,z) as above. It $\mathcal{V}(x,s) = b$ then output (s,b,z) as the simulated transcript. Else, try again.

This protocol has perfect completeness and soundness $1/2$. To amplify the soundness we simply repeat the entire protocol.

Remark. To maintain the zero-knowledge property we need to repeat the protocol *sequentially*. In general parallel repetition does not preserve ZK (see HW 4). But sequential repetition does!

If all we want is honest-verifier ZK then we can repeat the protocol in parallel.

References

[1] 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.