# *Zero-knowledge interactive proofs*

*MIT - 6.5620*
*Lecture 17 (Nov 9, 2026)*

> ***Warning:*** This document is a rough draft and may contain bugs. Please feel free to email us with corrections.

## *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 that this output is indeed correct? This is the topic for the next three 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 $T^\epsilon$ or even polylog$(T)$)? Unfortunately, we do not believe that every $T$-computable language has a proof (or a "witness") of size $<< T$. We overcome this barrier by changing the

definition of a proof and by making use of some cryptographic tools.

## *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 $R(x,w) = 1$.

Traditionally, 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.

A classical proof may reveal more than the truth of a statement: for example, a proof of graph isomorphism reveals an isomorphism, a proof of Hamiltonicity reveals a cycle, and a proof that $y$ is a quadratic residue modulo $N$ may reveal a square root.

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. In the plain noninteractive setting, such a requirement would limit us to languages in BPP.

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 the common input, previous messages, and the sender's private randomness. 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 $x$. 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 $\lambda$ times and accepting if and only if at least $\frac{\lambda}{2}$ are accepting we can get completeness $1 - \text{negl}(\lambda)$ and soundness $\text{negl}(\lambda)$ (because for independent Bernoulli trials $X_1, \ldots, X_\lambda$ with $\Pr[X_i = 1] = p$, the Hoeffding–Chernoff bound gives $\Pr[|\frac{1}{\lambda}\sum_{i=1}^{\lambda} X_i - p| > \delta] \leq 2e^{-2\delta^2\lambda}$. For adaptive cheating provers, the analogous soundness bound follows from martingale concentration.)

See this for information about the Chernoff bound, or the notes from the probability review recitation.

We therefore often give $\mathcal{P}$ and $\mathcal{V}$ an additional input $1^\lambda$, 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

## *Definition of Zero-Knowledge*

**Definition 3** (Honest verifier ZK)**.** Fix a language $L \in$ 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 $w$).

In class we saw definitions without the auxiliary input $z$, these definitions are slightly more general.

**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 $w$).

*Perfect, statistical, and computational ZK.* The definitions above require *computational* indistinguishability. Requiring the real and simulated views to have identical distributions gives *perfect* ZK; requiring negligible statistical distance gives *statistical* ZK. The QR and graph-isomorphism protocols below have perfect simulation with an *expected*-polynomial-time rewinding simulator. If a strict polynomial running-time bound is required, truncating the simulator yields statistical ZK instead.

## *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

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

Given an instance $(N, y)$ (with $y \in \mathbb{Z}_N^*$; otherwise the verifier rejects), 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. The verifier first checks $s, z \in \mathbb{Z}_N^*$, then accepts iff $z^2 = s \cdot y^b \pmod N$.

*Completeness and soundness.* The honest prover always passes the check. For soundness, if a cheating prover could answer both challenges for the same $s$, there would exist $z_0, z_1 \in \mathbb{Z}_N^*$ with $z_0^2 = s$ and $z_1^2 = sy \pmod N$. Hence $(z_1 z_0^{-1})^2 = y \pmod N$, contradicting $(N, y) \notin \mathbb{QR}$. Thus the soundness error is at most $1/2$.

*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 $s = z^2 \cdot y^{-1} \bmod N$.

The protocol is also (malicious verifier) ZK. The simulator is very similar to the one above: it guesses the (malicious) verifier's bit $b \leftarrow \{0, 1\}$, and produces $(s, z)$ as above. It samples and fixes the random coins of $\mathcal{V}^*$, and runs it on $s$. If its challenge equals the guessed $b$, it outputs $(s, b, z)$ (together with the verifier's coins); otherwise it rewinds and retries, keeping the verifier's coins fixed. Since the distribution of $s$ is independent of the guessed $b$, each attempt succeeds with probability $1/2$. Thus the expected number of attempts is two, and the simulated view is identically distributed to the real view.

To amplify the soundness from $1/2$ to negligible, we 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, but sequential repetition does. However, if all we want is honest-verifier ZK then we can repeat the protocol in parallel.

*Zero knowledge for Graph Isomorphism.* A second ZK protocol we saw in class is for the language $L_{\mathsf{iso}} = \{(G, H) : G \cong H\}$. Suppose

the prover knows an isomorphism $\pi$ with $H = \pi(G)$. The prover sends $K = \rho(G)$ for a uniformly random permutation $\rho$; the verifier responds with a random bit $b$. If $b = 0$, the prover reveals $\rho$, and the verifier checks $K = \rho(G)$. If $b = 1$, the prover reveals $\pi \circ \rho^{-1}$, and the verifier checks $H = (\pi \circ \rho^{-1})(K)$. Completeness is perfect. If $G$ and $H$ are not isomorphic, no fixed $K$ can pass both challenges, so soundness is $1/2$. To simulate, guess $b$: if $b = 0$, choose a random $\sigma$ and prepare $(K, \sigma) = (\sigma(G), \sigma)$; if $b = 1$, prepare $(K, \sigma^{-1}) = (\sigma(H), \sigma^{-1})$. Rewind whenever a malicious verifier chooses the other bit. As with QR, the simulated view is identical to the real one, and the expected number of attempts is two.

## *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.