Lecture 10: Verification in ML

Lecturer: Shafi Goldwasser **Date:** March 10, 2026
Scribes: Alaina Kolli, Akash Anand

1. ML as a Service

Let's consider a model where there is a client interacting with a service provider. The client can be any human or entity that provides data to a service provider, where the data is used to train a model. The service provider, such as OpenAI, Amazon, or Microsoft, trains the model on the data and returns it to the client. This setup requires the client to place complete trust in the service provider. However, it's possible that the service provider is adversarial to varying degrees of severity.

Given this hesitation, we want to trust the service provider but also verify the model returned to us. Specifically, we want to verify certain properties of the model after development. It is possible to incorporate verification during development as well, but currently, the norm is to perform verification post-development.

Let the model returned to us be h. Specifically, we want to verify the following properties of h:

- **Accuracy** - On average, h performs well on some benchmark.
- **Correctness** - We want to understand when h is correct and when it is not (e.g. hallucinations).
- **Robustness** - If the distribution shifts from what h was trained on, does the model still perform well? Is it generalizable?
- **Fairness** - h does not exhibit undue bias.
- **Safety** - A broader concept that includes alignment, robustness, etc.
- **Satisfies Regulations** - h satisfies any applicable regulations.

We want to be able to verify these properties without retraining the model ourselves; otherwise, what was the point of sending our data to a separate entity? Also, we may not have the power, time, or hardware to do so. Instead, we want to verify properties of h **cheaply**. In our case, cheaply can mean using:

- Fewer data samples
- Lower quality data
- Less time
- Less memory

- Black-box access to h only - no weights or internal details, only the ability to query

Our main goal is to be able to query the black-box model h and obtain guarantees about the properties listed above. This delegation-and-verification paradigm connects to the theory of verifiable computation [15, 19], and more recently to zero-knowledge proofs applied to ML inference [9, 26, 28].

2. Efficient Verification: from 80s Theory to Blockchains

Key Models and Their Origins

- Interactive Proofs & Arguments [3, 20] ~ 1985
- Zero-Knowledge Interactive Proofs [20] ~ 1985
- Non-Interactive Zero-Knowledge Proofs & Arguments, SNARKs, SNARGs [6, 7, 11] ~ 1987
- Multi-Prover Interactive Proofs [5] ~ 1988
- Debates [10] ~ 1989
- Probabilistically Checkable Proofs [2] ~ 1992

Remark 2.1. These models were initially developed for cryptography, foundational questions in complexity theory, and general intellectual curiosity, but they provide a natural starting point for adapting verification techniques to AI settings. In particular, the connection between multi-prover interactive proofs and AI alignment through debate has been explored in [8, 22].

There will always be two parties, the **Prover** and the **Verifier**, as seen in Fig. 1. The Prover is whoever trained the model, and the Verifier is the client seeking to verify the properties mentioned above. This Prover-Verifier model has appeared throughout cryptography, complexity theory, and verifiable delegation of computation to the cloud [19], where a client wants to outsource a computation and verify the result using only black-box access. We now want to apply this model to AI settings.

3. Efficiently Verifiable Proofs

In classical proofs, the process typically involves a claim, axioms, a sequence of derivation steps, and a final conclusion. We now want to consider *efficiently verifiable* proofs.

The Prover sends a proof of a claim to the Verifier, who can then accept or reject it. This requires some transfer of information between the two parties - and as we will see, the nature of this communication (how many rounds, what is revealed) matters greatly. In our ML setting, the Prover is the service provider and the Verifier is the client or a program acting on the client's behalf.

We want proofs that are efficiently verifiable - that is, the Verifier should be able to check a proof in polynomial time. This is precisely the notion captured by the complexity class NP. We will not worry about the complexity of the Prover for now, even though the Prover may work much

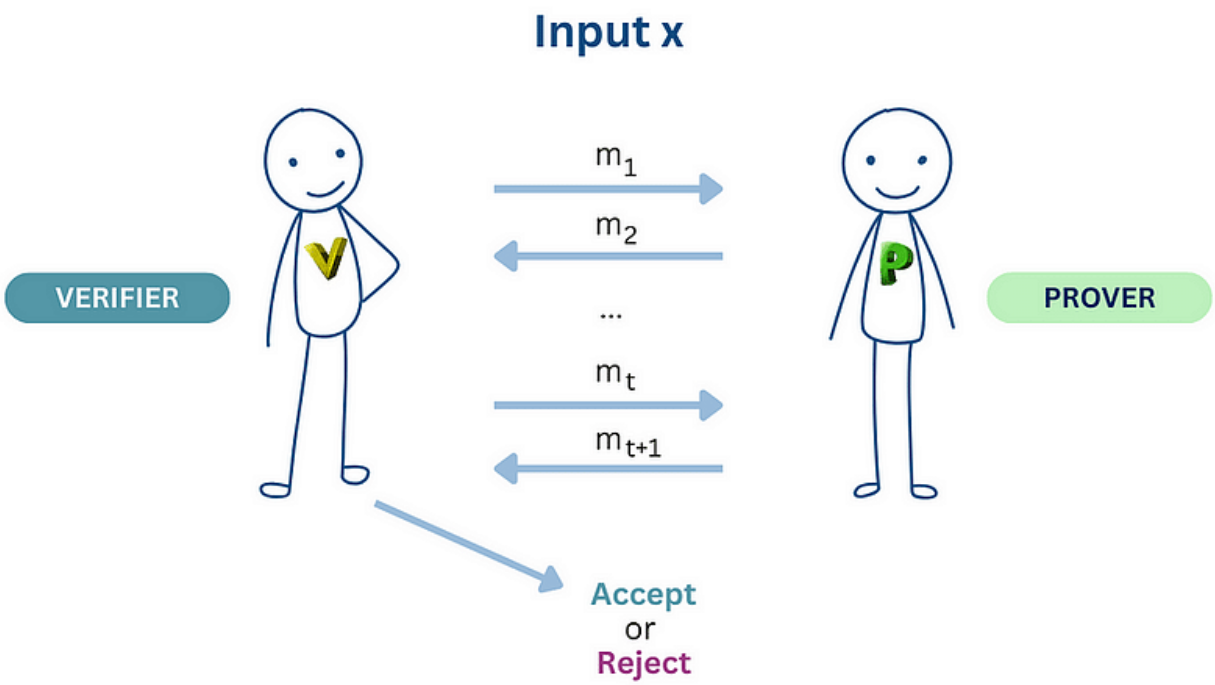

Figure 1: The Prover and Verifier receive a common input x and exchange messages $m_1, m_2, \ldots, m_{t+1}$. After the interaction, the Verifier outputs Accept or Reject. Image source: [30].

harder to produce the proof. The emphasis of this lecture is on the Verifier's ability to check proofs in polynomial time - much more efficiently than solving the problem from scratch without the Prover's help.

Factoring

Example 3.1 (Factoring)**.**
Claim 3.2. *N is a product of two large primes.*

Proof. The Prover is all-powerful so it can factor N. Hence, the Prover factors N, finds primes p and q, and sends them to the Verifier. The Verifier checks whether $p \cdot q = N$, which can be done in polynomial time. If the check passes, the Verifier accepts; otherwise it rejects.
Note: After the interaction, the Verifier knows not only that N is the product of two primes, but also the specific factors p and q. This additional information was not required to verify the claim. □

Quadratic Residue

Example 3.3 (Quadratic Residue)**.**
Claim 3.4. *y is a quadratic residue mod N (i.e., $y \equiv x^2 \pmod N$ for some x).*

Proof. The Prover sends x to the Verifier. The Verifier computes $x^2 \bmod N$ and accepts if the result equals y; otherwise it rejects.
Note: After the interaction, the Verifier knows not only that y is a quadratic residue mod N, but also a square root of y modulo N. This additional information was not required to verify the claim. □

Graph Isomorphism

Example 3.5 (Graph Isomorphism)**.**
Claim 3.6. *Graphs G_0 and G_1 are isomorphic.*

Proof. The Prover sends a permutation π mapping vertices of G_0 to vertices of G_1. The Verifier checks that for every pair of vertices i, j, there is an edge $(i, j) \in G_0$ if and only if $(\pi(i), \pi(j)) \in G_1$. This requires $O(n^2)$ checks, so verification runs in polynomial time.
Note: After the interaction, the Verifier knows not only that G_0 and G_1 are isomorphic, but also the specific isomorphism π. This additional information was not required to verify the claim. □

These problems are all in NP: for each one, the Prover sends a polynomial-length witness (a factorization, a square root, an isomorphism), and the Verifier checks it deterministically in polynomial time. Each such problem defines a *language* - the set of inputs for which a valid witness exists.

Definition 3.1 (NP Language)**.** L is an *NP-language* if there exists a polynomial-time verifier V such that

1. **Completeness:** For all $x \in L$, there exists a $\text{poly}(|x|)$-length witness $w \in \{0,1\}^*$ such that $V(x, w) = 1$.
2. **Soundness:** For all $x \notin L$ and all $\text{poly}(|x|)$-length $w \in \{0,1\}^*$, $V(x, w) = 0$.

For example, the set of all isomorphic graph pairs forms a language, as does the set of all pairs (N, y) such that $y \equiv x^2 \pmod N$ for some x. Soundness captures the guarantee that no false statement can be proven.

Remark 3.7. Some problems in NP are **NP-complete**, meaning that every problem in NP reduces to them in polynomial time [23]. A classic NP-complete problem is the Hamiltonian cycle problem; see [14] for additional background on NP-completeness.

Hamiltonian Cycle (NP-Complete)

Example 3.8 (Hamiltonian Cycle (NP-Complete))**.**
Claim 3.9. *Graph G has a Hamiltonian cycle.*

Proof. A Hamiltonian cycle visits every vertex exactly once and returns to the starting vertex. The Prover sends the Hamiltonian cycle to the Verifier, who checks that every vertex appears exactly once and that each consecutive pair of vertices is connected by an edge. This problem is NP-complete [23], meaning every problem in NP can be reduced to it.
Note: After the interaction, the Verifier knows not only that G has a Hamiltonian cycle, but also the cycle itself. This additional information was not required to verify the claim. □

This raises the question: *is there any other way to convince the Verifier without revealing the witness?*

4. Zero-Knowledge Proofs

The main idea is for the Prover to convince the Verifier that she *could* prove the claim, without actually revealing the proof itself. This concept was introduced in [20]. We now add two new ingredients to our proof model: non-trivial interaction and randomness.

Fact 4.1. ***Non-trivial interaction:*** *Rather than reading a static proof, the Verifier engages in a non-trivial interaction with the Prover. Back and forth interactions are allowed and the number of rounds of interaction will be bounded, but is more than a single message.*

Fact 4.2. ***Randomness:*** *The Verifier is randomized (tosses coins as a primitive operation) and can err with some small probability.*

Completeness stays the same, but soundness is affected by the addition of randomness. In NP, if $x \notin L$, no witness can make the Verifier accept. However, with randomness, even when $x \notin L$, there is a small probability - taken over the Verifier's coin tosses - that the Verifier will mistakenly accept.

The idea: Rubik's Cube Intuition

Suppose Alice the Prover wants to prove that there is a way to solve a Rubik's cube in $\leq k$ steps. One way is to send all k steps to the Verifier, who applies them and accepts if the cube is solved.
However, there is another approach. The Prover picks one of the many possible intermediate configurations at the $k/2$ mark and sends it to the Verifier. The Verifier then tosses a coin and issues one of two challenges:

- **Challenge 0:** Show me the first $k/2$ moves from the start to the intermediate configuration.
- **Challenge 1:** Show me the last $k/2$ moves from the intermediate configuration to the solved cube.

The Verifier asks to see either the first half or the second half, not both. If the Prover can consistently answer *both* types of challenges (across repeated rounds), then there exist k moves that solve the cube. This gives the Verifier probabilistic conviction without seeing the full solution.

Definition 4.1 (Interactive Proofs for Language Membership (Goldwasser, Micali, Rackoff [20]))**.** (P, V) is an interactive proof system for language L if

1. **Completeness:** If $x \in L$, then $\Pr[(P, V)[x] = \text{accept}] = 1$.
2. **Soundness:** If $x \notin L$, then for every (possibly cheating) prover strategy P^*,

$$\Pr[(P^*, V)[x] = \text{accept}] \leq \text{negl}(|x|),$$

where $\text{negl}(\cdot)$ denotes a negligible function (smaller than the inverse of any polynomial for sufficiently large input).

Remark 4.3. This definition adapts the NP definition - where soundness was perfect (no false statement is ever accepted) - to allow a small probability of error. Note that this is a one-sided definition: if the Verifier accepts, the statement is true (with high confidence), but if the Verifier rejects, it does not mean $x \notin L$ - it simply means the Verifier was not convinced.

Example 4.4. In the graph isomorphism example, how could we prove that two graphs are *not* isomorphic? The Prover would need to show that no permutation of vertices maps one graph to the other, but this would require the Prover demonstrating that all $n!$ possible permutations fail (far too many for a polynomial length witness).

In classical NP proofs, we can only prove "yes" instances - for example, we can prove two graphs are isomorphic by exhibiting the mapping, but we cannot efficiently prove they are *not* isomorphic, as that would require ruling out all $n!$ permutations. By introducing interaction and allowing a small probability of error, the class of statements a Prover can convince a Verifier of grows enormously. The class IP - the set of all languages with interactive proof systems - turns out to equal PSPACE, the class of problems solvable in polynomial space. PSPACE is a very large class: it contains all of NP and co-NP, and a polynomial-space machine can run for exponential time by reusing space [1]. This means that for any problem in PSPACE, the Prover can convince the Verifier of **both** yes and no instances.

Remark 4.5. Formally, the Prover and the Verifier are a pair of algorithms with a common input x. Each may also have its own private input. The distinction between public and private coins for the Verifier can be further explored in [18].

Remark 4.6. The definition above uses perfect completeness (the Verifier always accepts when $x \in L$) and negligible soundness error, but more generally we can define a completeness threshold $c(x)$ and a soundness threshold $s(x)$ such that: if $x \in L$, then $\Pr[(P,V)[x] = \text{accept}] > c(x)$, and if $x \notin L$, then for all P^*, $\Pr[(P^*,V)[x] = \text{accept}] < s(x)$.

Remark 4.7. With completeness, more work tends to be required to achieve perfect completeness from the threshold $c(x)$. The soundness error can be reduced by repeating the protocol independently many times and taking the majority outcome. For example, if the soundness error is $\frac{1}{3}$, repeating k times drives the error exponentially close to 0 (making it satisfy the negligible constraint).

Class IP

Just as NP is the class of languages with efficiently verifiable proofs, we define IP as the class of languages with interactive proof systems:

$$\mathbf{IP} = \left\{ L \;\middle|\; \exists\, (P,V) \text{ interactive proof system for } L \text{ with completeness } c(x) \geq \tfrac{2}{3} \text{ and soundness } s(x) \leq \tfrac{1}{3} \right\}.$$

Simply put, this is the set of languages where there exists an interactive proof system that has the properties of completeness and soundness. It is known that $\mathbf{NP} \subseteq \mathbf{IP}$ [20] and, remarkably, that $\mathbf{IP} = \mathbf{PSPACE}$. The result $\mathbf{IP} = \mathbf{PSPACE}$ was proved by Shamir [27], building on the algebraic techniques of Lund, Fortnow, Karloff, and Nisan [24].

Interactive Proofs for Isomorphism

The **goal** is for the Prover to convince the Verifier that G_0 and G_1 are isomorphic, without revealing the isomorphism itself.

Recall that G_0 and G_1 are isomorphic if there exists a permutation σ of vertices such that $(i, j) \in E_0$ if and only if $(\sigma(i), \sigma(j)) \in E_1$.

Protocol:

1. Alice the Prover picks a random bit $r \in \{0, 1\}$ and a random permutation γ, and sends $H = \gamma(G_r)$ to Bob.
2. Bob the Verifier picks a random bit $b \in \{0, 1\}$ and sends it to Alice.
3. Alice must respond with an isomorphism from G_b to H.
   - If $b = r$: Alice sends γ.
   - If $b \neq r$: Alice uses her knowledge of the isomorphism σ between G_0 and G_1 to compose and produce the correct mapping.
4. Bob verifies that the permutation Alice sent is indeed an isomorphism from G_b to H.

Remark 4.8. The case $b \neq r$ works because if G_0 and G_1 are truly isomorphic via $\sigma : G_0 \to G_1$, then H is isomorphic to both. For example, if $r = 0$ and $b = 1$, then $H = \gamma(G_0)$ and Alice needs an isomorphism from G_1 to H. She sends $\gamma \circ \sigma^{-1}$, which maps $G_1 \to G_0 \to H$. The case $r = 1, b = 0$ is symmetric.

Assuming the Prover knows the isomorphism σ between G_0 and G_1, completeness and soundness hold:

Completeness: If G_0 and G_1 are isomorphic, Alice can produce a valid isomorphism from G_b to H for either value of b.

Soundness: If G_0 and G_1 are not isomorphic, then H is isomorphic to at most one of G_0, G_1. The best a cheating Alice can do is guess $b = r$, which succeeds with probability $\frac{1}{2}$. Repeating k times independently reduces the soundness error to $\left(\frac{1}{2}\right)^k$.

Remark 4.9. A fresh H must be chosen each round. If Alice reused the same H, Bob could ask different questions across rounds and learn isomorphisms from *both* G_0 and G_1 to H, thereby recovering the isomorphism between G_0 and G_1.

Interactive Proofs for QR (Quadratic Residuosity)

Define the language $L_{\text{QR}} = \{(N, y) \mid \exists\, x \in \mathbb{Z}_N^* \text{ s.t. } y \equiv x^2 \pmod N\}$. That is, L_{QR} is the set of pairs (N, y) where y is a quadratic residue mod N.

Goal: Convince the Verifier that $(N, y) \in L_{\text{QR}}$ without revealing x such that $y \equiv x^2 \pmod N$.

Protocol:

1. Alice the Prover picks a random $r \in \mathbb{Z}_N^*$ and sends $s = r^2 \bmod N$ to Bob the Verifier.
2. Bob picks a random bit $b \in \{0, 1\}$ and sends it to Alice.
3. Alice sends:
$$z = \begin{cases} r & \text{if } b = 0 \\ r \cdot x \bmod N & \text{if } b = 1 \end{cases}$$
4. Bob accepts iff $z^2 \equiv s \cdot y^b \pmod N$.

Note that in both cases, $z = r \cdot x^b \bmod N$. Alice must know x (which is fixed) to respond to challenge $b = 1$, while r is freshly chosen each round. Bob simply checks whether $z^2 \equiv s \cdot y^b \pmod N$.

Claim 4.10. *If $(N, y) \in L_{\text{QR}}$, then the Verifier accepts with probability 1.*

Proof. Since $y \equiv x^2 \pmod N$, we have $z^2 = (r \cdot x^b)^2 = r^2 \cdot (x^2)^b \equiv s \cdot y^b \pmod N$. So the Verifier's check passes for both values of b. □

Claim 4.11. *If $(N, y) \notin L_{\text{QR}}$, then for every cheating prover P^*, the Verifier accepts with probability at most $\frac{1}{2}$.*

Proof. Suppose the Verifier accepts with probability $> \frac{1}{2}$. Since b is a single random bit, the prover must be able to answer both challenges for some fixed s. That is, there exist $z_0, z_1 \in \mathbb{Z}_N^*$ such that $z_0^2 \equiv s \pmod N$ and $z_1^2 \equiv s \cdot y \pmod N$. Then $(z_1/z_0)^2 \equiv y \pmod N$, meaning z_1/z_0 is a square root of y mod N, contradicting $(N, y) \notin L_{\text{QR}}$. □

As before, repeating this protocol k times independently reduces the soundness error to $\left(\frac{1}{2}\right)^k$.

5. Zero-Knowledge Interactive Proofs

The key idea of zero-knowledge is that the Verifier learns nothing from the interaction except that the statement is true. After the interactive proof, the Verifier has a *view* of the interaction: the transcript of messages exchanged and the coins the Verifier tossed. A proof is zero-knowledge if this view gives the Verifier nothing beyond what it could have computed on its own, without interacting with the Prover. See [16] for more explanations.

In contrast, in the NP setting, the Verifier receives the actual witness (e.g., the factorization, the square root, the isomorphism), which is information beyond what is needed to verify the claim.

How do we capture getting "nothing extra"?

An interactive proof is zero-knowledge if there exists an efficient algorithm (a *simulator*) that, without interacting with the Prover, can produce a view that is indistinguishable from the Verifier's real view of the interaction. If such a simulator exists, the Verifier could have produced the same view on its own, meaning it learned nothing beyond the truth of the statement.

Definition 5.1 (Zero-Knowledge Interactive Proof)**.** An interactive protocol (P, V) is *honest-verifier zero-knowledge* for a language L if there exists a PPT algorithm S (a simulator) such that for every $x \in L$, the following two distributions are indistinguishable:

1. $\mathsf{view}_V[(P, V)(x)]$ (the real interaction)
2. $S(x, 1^k)$ (the simulated view, where k is the security parameter)

(P, V) is a *zero-knowledge interactive proof* if it is complete, sound, and zero-knowledge. Note that it is not required that these distributions be the same when $x \notin L$.

Remark 5.1. This definition assumes an honest Verifier that follows the protocol. We will strengthen this to handle malicious verifiers later.

Definition 5.2 (Perfect Zero-Knowledge Interactive Proof)**.** An interactive protocol (P, V) is perfect zero-knowledge for a language L if there exists a simulator S such that for every $x \in L$, the following two distributions are *identical* (not merely indistinguishable):

$$\mathsf{view}_V[(P, V)(x)] \equiv S(x, 1^k).$$

(P, V) is a *honest verifier perfect zero-knowledge interactive proof* if it is complete, sound, and perfect zero-knowledge. Note that S is allowed to run in *expected* polynomial time (rather than strict polynomial time).

Example 5.2 (Graph Isomorphism: Perfect Zero-Knowledge)**.** Following the protocol above, the Prover chooses a random permutation γ_0 and sets $H = \gamma_0(G_0)$, then sends H to the Verifier. The Verifier tosses a coin $b \in \{0, 1\}$ and sends it to the Prover, who responds with:

- If $b = 0$: the permutation γ_0 (an isomorphism from G_0 to H).
- If $b = 1$: the permutation $\gamma_0 \circ \sigma^{-1}$ (an isomorphism from G_1 to H, where $\sigma : G_0 \to G_1$ is the isomorphism the Prover knows).

The Verifier's view consists of $(H,\ b,\ \gamma'_b)$, where γ'_b is a random isomorphism from G_b to H.

We construct a simulator S that produces the same view without interacting with the Prover:

1. Toss a coin $b' \in \{0, 1\}$.
2. Choose a random permutation $\gamma_{b'}$ and set $H = \gamma_{b'}(G_{b'})$.
3. Output the simulated transcript $(H,\ b',\ \gamma_{b'})$.

The simulated view is identically distributed to the real view: in both cases, H is a random permutation of $G_{b'}$, the coin b' is uniformly random, and $\gamma_{b'}$ is a random isomorphism from $G_{b'}$ to

H. The only difference is the order in which these values are generated, which does not affect the distribution.

The simulation above assumes an *honest* Verifier - one that follows the protocol and chooses its challenges by tossing fair coins. However, a *malicious* Verifier V^* might choose its messages adaptively in an attempt to extract additional information from the Prover. To handle this, we need a stronger definition of zero-knowledge.

Definition 5.3 (Zero-Knowledge Against Malicious Verifiers)**.** An interactive protocol (P, V) is *perfect zero-knowledge against malicious verifiers* for a language L if there exists an expected polynomial-time simulator S such that for every PPT verifier strategy V^* and every $x \in L$, the two distributions are identical:

$$\mathsf{view}_{V^*}[(P, V^*)(x)] \equiv S^{V^*}(x, 1^k),$$

where k is the security parameter and S^{V^*} denotes that S has oracle access to V^*.

Rewinding Technique

The honest-verifier simulator from before does not work against a malicious V^*, because we cannot predict what bit b the malicious Verifier will send. The rewinding technique handles this by guessing b and retrying if the guess is wrong.

Simulator S for malicious V^*:

1. Toss a coin $b' \in \{0, 1\}$.
2. Choose a random permutation $\gamma_{b'}$ and set $H = \gamma_{b'}(G_{b'})$.
3. Feed H to V^* and receive its challenge $b = V^*(H)$.
4. If $b = b'$: output $(H,\ b,\ \gamma_{b'})$ as the simulated transcript and stop.
5. If $b \neq b'$: **abort and restart from step 1** ("rewind" V^*).

Claim 5.3. $\Pr[b = b'] = \frac{1}{2}$, *since b' is uniformly random and independent of V^*'s choice of b. Therefore, the expected number of attempts before success is 2. For k independent sequential repetitions of the full protocol, the expected total number of simulation attempts is $2k$.*

The simulator runs in *expected* polynomial time (not worst-case), which is acceptable for zero-knowledge.

Theorem 5.4. *The QR protocol is (malicious-verifier) perfect zero-knowledge.*

Proof. **Simulator S for QR:**

1. Toss a coin $b' \in \{0, 1\}$.
2. Choose a random $r \in \mathbb{Z}_N^*$ and set $s = r^2 \cdot y^{-b'} \bmod N$.
3. Feed s to V^* and receive $b = V^*(s)$.
4. If $b = b'$: output $(s,\ b,\ r)$ as the simulated transcript and stop.
5. If $b \neq b'$: abort and restart from step 1.

When the simulator succeeds, the output (s, b, r) satisfies $r^2 = s \cdot y^b \pmod N$ by construction (since $b = b'$). This is identically distributed to a real execution, where the Prover sends $s = r^2$,

receives b, and responds with $z = r \cdot x^b$. The simulator succeeds with probability $\frac{1}{2}$ per attempt, so it runs in expected polynomial time. □

Lemma 5.5. *The simulator S in the proof of Theorem 5.4 runs in expected polynomial time and its output is identically distributed to the view of V^* in a real execution.*

What Made Zero-Knowledge Proofs Possible?

Randomness:

- The statement to be proven has many possible proofs, and the Prover chooses one at random.
- Each proof is made up of two parts: seeing one part alone reveals no knowledge, but being able to provide both parts guarantees the statement is true.
- The Verifier randomly chooses which of the two parts the Prover must reveal. The Prover's ability to provide either part convinces the Verifier.

The random self-reducibility properties underlying these protocols were studied more generally by Tompa and Woll [29].

6. Zero-Knowledge Proofs of Knowledge

In the protocols above, the Prover appears to have demonstrated more than just the truth of a statement - she has shown that she actually "knows" a witness (e.g., the isomorphism, the square root).

Definition 6.1 (Proof of Knowledge [4])**.** Let V be a polynomial-time relation. Let $(x, w) \in V$. V defines the language $L_V = \{x \mid \exists\, w \text{ s.t. } V(x, w) = 1\}$. We say that (P, V) is a **proof of knowledge** for L_V [or that P on x knows w] if there exists an extractor algorithm E such that: for any (possibly cheating) prover P^* that convinces V to accept with non-negligible probability $\epsilon(n)$, $E^{P^*}(x)$ outputs w in expected polynomial time $\text{poly}(n, 1/\epsilon(n))$.

$E^{P^*}(x)$: E can run P^* on the same randomness repeatedly, asking P^* different questions in multiple executions.

A protocol that is simultaneously zero-knowledge and a proof of knowledge is called a **Zero-Knowledge Proof of Knowledge (ZKPOK)**. See also [13] for early formulations of this concept.

Example 6.1 (ZKPOK for Graph Isomorphism)**. Extractor E:**

1. Run the protocol, receiving graph H from P^*. Save the state of P^*.
2. Issue challenge $b = 0$; receive isomorphism $\gamma_0 : G_0 \to H$.
3. Rewind P^* to the saved state and issue challenge $b = 1$; receive isomorphism $\gamma_1 : G_1 \to H$.
4. Output $\sigma = \gamma_1^{-1} \circ \gamma_0$, which is an isomorphism from G_0 to G_1.

Remark 6.2. The first applications of zero-knowledge proofs included preventing identity theft, proving properties of secrets without revealing them, verifying statements beyond what classical NP proofs can handle efficiently, and constructing secure protocols [12].

Identity Theft Example

In classical authentication, the user sends a password over the network, where it can be intercepted. With zero-knowledge, the Prover can convince the Verifier that she knows the password (or secret key) without ever transmitting it. The Verifier becomes convinced of the Prover's identity but learns nothing that would allow it to impersonate the Prover. The Fiat-Shamir identification scheme [12] was the first practical realization of this idea.

Definition 6.2 (Computational Zero-Knowledge)**.** An interactive protocol (P, V) is *computationally zero-knowledge* for a language L if for every PPT V^*, there exists a PPT simulator S such that for every $x \in L$, the following two distributions are *computationally indistinguishable* (no efficient algorithm can distinguish them):

$$\mathsf{view}_{V^*}[(P, V^*)(x)] \approx_c S(x, 1^k),$$

where k is the security parameter. (P, V) is a *computationally zero-knowledge interactive proof* if it is complete, sound, and computationally zero-knowledge. Note that this relaxes perfect zero-knowledge, where the distributions must be identical, to requiring only that no efficient distinguisher can tell them apart.

Remark 6.3. The simulator S may arbitrarily depend on V^* (rather than having oracle access to it). Both formulations are equivalent, and the choice depends on the context.

7. Zero-Knowledge for All of NP

The following result shows that zero-knowledge proofs are not limited to specific problems like quadratic residuosity or graph isomorphism, but extend to all of NP.

Theorem 7.1 (Goldreich, Micali, Wigderson 1986 [17])**.** *If one-way functions exist, then every problem in NP has a computational zero-knowledge interactive proof.*

One-Way Functions

A function $f : \{0,1\}^* \to \{0,1\}^*$ is a *one-way function* if:

1. f is computable in polynomial time.
2. For every PPT adversary A and all sufficiently large n:

$$\Pr_{x \leftarrow \{0,1\}^n}\left[A(f(x)) \in f^{-1}(f(x))\right] \le \mathsf{negl}(n).$$

One-way functions are essential for complexity-based cryptography [21]. Intuitively, f is easy to compute but hard to invert. One-way functions imply *commitment schemes* [25], which are the key building block for the construction below.

To prove the theorem, it suffices to give a zero-knowledge interactive proof for a single NP-complete problem, since any problem in NP can be reduced to it. We use 3COLOR, the 3-colorability problem.

Definition 7.1 (3-Colorability)**.** $\mathsf{3COLOR} = \{G = (V, E) \mid \exists\, \chi : V \to \{0,1,2\} \text{ s.t. } \forall (u,v) \in E,\ \chi(u) \neq \chi(v)\}$.

Since 3COLOR is NP-complete, for any $L \in \mathbf{NP}$ there is a polynomial-time reduction mapping instances x of L to graphs G_x such that $x \in L \iff G_x \in$ 3COLOR. A zero-knowledge proof for 3COLOR thus yields one for all of NP.

Zero-Knowledge Proof for 3-Coloring

Common input: Graph $G = (V, E)$.
Prover's private input: A valid 3-coloring $\chi : V \to \{0, 1, 2\}$.
Protocol (one round):

1. **Commit:** The Prover picks a random permutation σ of $\{0, 1, 2\}$ and applies it to the coloring, obtaining $\alpha = \sigma \circ \chi$. For each vertex $v \in V$, the Prover commits to $\alpha(v)$ using a commitment scheme [25]. The Prover sends all $|V|$ commitments to the Verifier.
2. **Challenge:** The Verifier picks a random edge $(u, v) \in E$ and sends it to the Prover.
3. **Response:** The Prover opens the commitments for u and v, revealing $\alpha(u)$ and $\alpha(v)$.
4. **Verification:** The Verifier accepts iff $\alpha(u) \neq \alpha(v)$.

Completeness: A valid coloring always satisfies $\alpha(u) \neq \alpha(v)$ for all edges, so the Verifier always accepts.
Soundness: If G is not 3-colorable, then any coloring must have at least one monochromatic edge. The Verifier catches a cheating Prover with probability $\geq \frac{1}{|E|}$ per round. This uses the *binding property* of commitments: once the Prover commits, they are bound to a unique color and cannot reveal $\alpha(u)$ and $\alpha(v)$ arbitrarily. Repeating $O(|E| \cdot k)$ times amplifies soundness.
Zero-knowledge: The Verifier sees only two colors on one random edge. Due to the random permutation σ, these are a uniformly random pair of distinct colors, independent of the actual coloring χ. A simulator can produce the same distribution without knowing χ, using the *hiding property* of commitments: the unopened commitments do not reveal any information about the hidden values.

Commitment Schemes

A commitment scheme is a two-phase cryptographic protocol where the Prover can "commit" to a value without revealing it, and later "open" the commitment to prove what was committed. Formally:

- **Hiding:** Before the opening phase, the commitment reveals no information about the committed value. Even a computationally powerful adversary cannot distinguish commitments to two different values.
- **Binding:** Once a commitment is made, the Prover cannot open it to two different values. The commitment cryptographically "binds" the Prover to a unique value.

Example 7.2 (Hamiltonian Cycle: Zero-Knowledge Proof (NP-Complete))**. Common input:** Graph $G = (V, E)$.
Prover's private input: A Hamiltonian cycle C in G.

Protocol:

1. The Prover picks a random permutation π of the vertex labels, constructs the permuted graph $G' = \pi(G)$, and *commits* to each possible edge (i, j) with $i \neq j$ as a bit (1 if an edge, 0

if not). The Prover sends all $O(n^2)$ commitments to the Verifier.

2. The Verifier picks a random bit $b \in \{0, 1\}$ and sends it to the Prover.
3. The Prover responds:
   - $b = 0$: Open all commitments and reveal π, showing that $G' = \pi(G)$.
   - $b = 1$: Open only the n commitments corresponding to the edges of the Hamiltonian cycle $\pi(C)$ in G'.
4. The Verifier accepts iff the revealed information is consistent with the claimed structure.

Completeness: If G has a Hamiltonian cycle C, the Prover can answer both challenges correctly.

Soundness: If G has no Hamiltonian cycle, a cheating Prover can satisfy at most one of the two challenges. The Verifier catches the cheat with probability $\frac{1}{2}$ per round. Repeating k times gives soundness error $\left(\frac{1}{2}\right)^k$.

Zero-knowledge: This protocol is computationally zero-knowledge. The simulator guesses the Verifier's challenge $b' \in \{0, 1\}$ in advance and prepares accordingly. If $b' = 0$: the simulator commits to $\pi(G)$ for a random π (which it can do without knowing C). If $b' = 1$: the simulator commits to a random Hamiltonian cycle on n vertices embedded in an otherwise empty graph. If the Verifier's actual challenge $b \neq b'$, the simulator rewinds and tries again. When $b = b'$, the hiding property of the commitment scheme ensures the unopened commitments reveal nothing, so the simulated view is computationally indistinguishable from the real view.

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[3] László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429, 1985.

[4] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Annual International Cryptology Conference*, pages 390–420. Springer, 1992.

[5] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 373–410. 2019.

[6] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 326–349, 2012.

[7] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, pages 329–349. 2019.

[8] Jonah Brown-Cohen, Geoffrey Irving, and Georgios Piliouras. Scalable ai safety via doubly-efficient debate. *arXiv preprint arXiv:2311.14125*, 2023.

[9] Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574, 2024.

[10] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter Shor. Probabilistically checkable debate systems and approximation algorithms for pspace-hard functions. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, pages 305–314, 1993.

[11] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 52–72. Springer, 1987.

[12] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[13] Uriel Fiege, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 210–217, 1987.

[14] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[15] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

[16] Oded Goldreich. *Foundations of cryptography, volume 2*. Cambridge university press Cambridge, 2004.

[17] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38 (3):690–728, 1991.

[18] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68, 1986.

[19] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.

[20] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, pages 203–225. 2019.

[21] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *30th Annual Symposium on Foundations of Computer Science*, pages 230–235. IEEE Computer Society, 1989.

[22] Geoffrey Irving, Paul Christiano, and Dario Amodei. Ai safety via debate. *arXiv preprint arXiv:1805.00899*, 2018.

[23] Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009.

[24] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.

[25] Moni Naor. Bit commitment using pseudorandomness. *Journal of cryptology*, 4(2):151–158, 1991.

[26] Zhizhi Peng, Taotao Wang, Chonghe Zhao, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. A survey of zero-knowledge proof based verifiable machine learning. *arXiv preprint arXiv:2502.18535*, 2025.

[27] Adi Shamir. Ip= pspace. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

[28] Haochen Sun, Jason Li, and Hongyang Zhang. zkllm: Zero knowledge proofs for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 4405–4419, 2024.

[29] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 472–482. IEEE, 1987.

[30] Veridise. ZK fundamentals: Proof systems. https://veridise.com/blog/learn-blockchain/zk-fundamentals-proof-systems/, 2024. Accessed: March 16, 2026.