

Zero Knowledge Draft Lecture Notes 2026

1 Introduction to Zero Knowledge

The obvious interactive proof protocol for GI is for P to simply send V the isomorphism between G_0 and G_1 . This corresponds to the traditional, non-interactive way of proving things. However, this protocol has the undesirable feature of revealing to V much information. In particular, V now knows an isomorphism between G_0 and G_1 . We desire an interactive proof which still convinces V that the graphs are isomorphic without revealing so much information to V . In fact, we don't want P to reveal anything to V beyond that the graphs are isomorphic. Such proofs are called *zero-knowledge*, introduced by Goldwasser, Micali and Rackoff (GMR-85). Protocol (P1) below is one such protocol for GI.

1.1 Motivating story

(This is a story also due to [GMR].) One night in a small town there is a murder. Verry Fire, the local reporter, hears about the murder and wants to write a story for her newspaper. She goes to a pay phone and calls the detective to get the facts of the murder case, but the detective simply tells her "There was a murder" and he hangs up. She calls back several times, but every time she just hears the phrase "There was a murder." Verry already knew that there was a murder, so she certainly didn't need to call the detective to obtain this information. She could have just saved her money and generated this phrase herself. Feeling a little frustrated, she decides to call the police chief. The chief, who enjoys playing games with reporters, flips a coin, and if the coin is heads, the chief says "There was a murder" and hangs up. If the coin is tails, he says "No comment" and hangs up. Verry calls back several times and sometimes she hears the first phrase while other times she hears the second phrase. Her conversation with the chief, however, still hasn't given her any new information for her column. She could have just flipped a coin herself and generated the chief's phrases with the same probability distribution. Verry proceeds to write her column, but she could have written the column without talking to the detective or the chief, because her conversations with them were zero-knowledge.

2 Introduction to Interactive Proofs

2.1 Introduction

A traditional, Euclidean-style proof for an assertion consists of a prover who simply outputs a proof. Someone reading the proof, a verifier, then decides whether or not the proof is correct. The observation has been made that there could be an advantage to letting the verifier *interact* with the prover. This may allow the assertion to be proven faster or with the release of less information than would be the case if the verifier were passive.

Our general framework consists of a prover P who is computationally unbounded, and a verifier V who is allowed only polynomial time. Both P and V are allowed to flip coins and they communicate with each other by sending messages. Note that since V is a polynomial-time machine, only a polynomial number of messages will be sent between P and V .

The programs for P and V define the protocol PV . The input, typically an assertion of the form $x \in L$, is presented to both P and V , and P tries to convince V that the assertion is true. If V is convinced, then V accepts.

2.2 Definition of IP [Goldwasser, Micali, Rackoff]

Definition 1. An Interactive Proof for a language L is a protocol PV for a Prover and a Verifier such that:

- Completeness: If $x \in L$ then P has a good chance of convincing V that $x \in L$:

$$\forall c > 0 \exists N \forall x \in L, |x| > N \implies \Pr[PV(x) \text{ makes } V \text{ accept}] > 1 - \frac{1}{|x|^c}$$

- Soundness: If $x \notin L$ then every P' has little chance of convincing V that $x \in L$:

$$\forall c > 0 \exists N \forall P' \forall x \notin L, |x| > N \implies \Pr[P'V(x) \text{ makes } V \text{ accept}] < \frac{1}{|x|^c}$$

The probabilities are over the coins of the participating machines. IP is defined to be the class of languages which have Interactive Proofs.

We first describe protocols with error $1/2$ in one trial. Repeating k times with the indicated acceptance rule reduces this error to 2^{-k} . Taking $k = |x|$, for example, makes the error smaller than every inverse polynomial.

2.3 IP for Graph Nonisomorphism

Disclaimer: In these notes we will write interactive proofs for two languages: graph isomorphism (GI) and graph nonisomorphism (GNI). These languages are chosen because, besides the belief that V (a PPT machine) cannot recognize them, they provide a convenient framework in which to study the notions of interactive proofs and zero knowledge. In addition, protocols for GI and GNI can be translated into protocols for certain hard number-theoretic problems. One would not really base a real system on GI or GNI.

Many of these interactive proofs will rely on the ability to produce random permutations of graphs. We do this by creating a random permutation and applying it to the graph. A random permutation gives a uniformly random labeled copy, including when a graph has symmetries. Whenever an isomorphism is chosen at random below, it is uniform among the isomorphisms between the specified graphs. All graphs in a protocol have the same labeled vertex set. Malformed messages are rejected, and an honest party stops if the other party stops.

$x \in GNI$ iff x is a pair of graphs (G_0, G_1) and $G_0 \not\sim G_1$. We abbreviate this as $x = \{G_0 \not\sim G_1\}$.

The following protocol is an interactive proof for GNI. V picks either G_0 or G_1 at random and generates a random permutation of that graph. V sends this new graph to P , who responds by telling V which graph V originally picked. Repeat this k times. V accepts if P is right every time. In tabular form:

$x = \{G_0 \not\sim G_1\}$		
P	communication	V
	$\xleftarrow{G'}$	Generate a random bit b and a random permutation Π . Let $G' = \Pi(G_b)$.
Determine b' such that $G' \sim G_{b'}$.	$\xrightarrow{b'}$	Reject if $b' \neq b$.
		Repeat k times. Accept if $b' = b$ every time.

If there is no unique such bit, the honest prover sends a failure symbol and this trial is rejected. This specifies its behavior on arbitrary verifier messages.

This protocol is an interactive proof because:

- **Completeness:** If the graphs are not isomorphic then only one of G_0 or G_1 will be isomorphic to G' , so P will always be able to determine b .
- **Soundness:** If the graphs are isomorphic then G' could have come from either G_0 or G_1 with equal probability. P can only guess at b with a $1/2$ chance of being right. Since the experiment is run k times, the probability that P guesses right every time is $1/2^k$.

2.4 Protocol (P1): Interactive Proof for Graph Isomorphism

$x \in GI$ iff x is a pair of graphs (G_0, G_1) and $G_0 \sim G_1$. We abbreviate this as $x = \{G_0 \sim G_1\}$.

The following protocol is an interactive proof for GI. P generates a random permutation of G_0 . P sends this new graph to V , who responds with a bit b . P then responds to V 's request by sending a permutation which maps G_b to the new graph that P generated. V checks that this permutation is actually an isomorphism between the graphs. Repeat this k times. V accepts if P was able to send a correct permutation every time. In tabular form:

$x = \{G_0 \sim G_1\}$		
P	communication	V
Generate a random permutation Π_1 . Let $G' = \Pi_1(G_0)$.	$\xrightarrow{G'}$	
	$\xleftarrow{b}$	Generate a random bit b .
Determine Π_2 such that $G' = \Pi_2(G_b)$.	$\xrightarrow{\Pi_2}$	Reject if Π_2 is not a permutation or $G' \neq \Pi_2(G_b)$.
		Repeat k times sequentially. Accept if every trial passes.

The prover can compute Π_2 efficiently if it knows an isomorphism between G_0 and G_1 : use Π_1 when $b = 0$, and compose Π_1 with a fixed isomorphism from G_1 to G_0 when $b = 1$.

This protocol is an interactive proof because:

- **Completeness:** If the graphs are isomorphic then G' is isomorphic to both G_0 and G_1 , so P can always send an isomorphism between G' and G_b .
- **Soundness:** If the graphs are not isomorphic then G' is isomorphic to at most one of G_0 and G_1 . Thus, for any graph sent by a cheating prover P' , at most one value of b can be answered correctly. The probability that P' can do this k times is at most $1/2^k$.

3 Definition of Zero Knowledge

Definition 2. For a protocol PV , let $PV(x)$ represent the view of the conversation from the verifier's point of view on input x . Specifically, $PV(x)$ consists of:

- The messages sent back and forth between P and V
- The coins of V

Early stopping and invalid messages are included in the view.

In a later section we will justify including the coins of V in this definition. Let $[PV(x)]$ represent the distribution of points $PV(x)$ taken over the coins of P and V . In general, for a machine S , let $[S(x)]$ be the distribution of outputs of S on input x taken over the coins of S .

Definition 3. An Interactive Proof PV for a language L is Zero Knowledge if

$$\forall V' \exists S_{V'} \text{ running in expected polynomial time such that } \forall x \in L$$

$$[S_{V'}(x)] \simeq [PV'(x)]$$

The quantifier over V' ranges over polynomial-time verifiers.

$S_{V'}$ knows V' and, on input $x \in L$, outputs points of the form $PV'(x)$ defined above. Intuitively, the existence of $S_{V'}$ shows that V' does not need P to generate the output distribution $[PV'(x)]$. V' could have generated the distribution itself in expected polynomial time. Therefore P does not transfer any knowledge to V' (beyond the fact that $x \in L$) which V' could not have generated itself.

The quantifier is over all verifiers V' , even cheating verifiers. That is, PV' doesn't have to be an interactive proof for L . The only goal of V' may be to extract information from P . Even for such cheating verifiers, there must exist a simulator with $\simeq$ output distribution.

The existence of a simulator for a verifier V' means that the interaction of P and V' is zero knowledge. If the interaction of P with every verifier V' is zero knowledge, then the protocol PV is zero knowledge. In this case, note that the protocol PV' for any verifier V' is also zero knowledge, but remember that this protocol is not necessarily an interactive proof that $x \in L$. The same requirement applies when the verifier has auxiliary information: give that same information to the simulator and include it in the view. We suppress it in the notation.

We use expected polynomial time for the simulators. A bound on the expectation is not a bound on every execution. As usual in these graph protocols, we count exact uniform sampling of a permutation as a polynomial-time operation. With fair random bits, this sampling takes expected polynomial time. A polynomial cutoff makes sampling strictly polynomial time and introduces only negligible additional error. The perfect statements below use exact sampling.

3.1 Perfect, statistical, and computational zero knowledge

In the definition of ZK, we use a simulator whose distribution on $x \in L$ is $\simeq$ to the distribution of $PV'(x)$. There are actually three different definitions of ZK corresponding to the three different definitions of $\simeq$ of distributions:

Definition 4. The three variants of ZK are:

- **Perfect ZK** ($=$). The distributions are exactly equal. This is the strictest definition of ZK.
- **Statistical ZK** ($\stackrel{s}{=}$). The distributions are statistically close. Distributions $\{X_n\}$ and $\{Y_n\}$ are statistically close iff

$$\forall c > 0 \exists N \forall n > N, \quad \frac{1}{2} \sum_{\alpha} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]| < \frac{1}{n^c}$$

The sum is over all possible outputs. Even a computationally unbounded machine has only negligible distinguishing advantage from one sample. A polynomial number of independent samples still gives only negligible advantage.

- **Computational ZK** ($\stackrel{c}{=}$). The distributions are computationally indistinguishable to polynomial-time machines. Distributions $\{X_n\}$ and $\{Y_n\}$ are polynomial-time indistinguishable iff

$$\forall c > 0 \forall A \in PPT \exists N \forall n > N, \quad |\Pr[A(X_n) = 1] - \Pr[A(Y_n) = 1]| < \frac{1}{n^c}$$

The machine A is also given n , and the probabilities include its coins. Thus no polynomial-time machine has a nonnegligible distinguishing advantage.

For verifier views, the bounds must hold uniformly over true inputs and auxiliary information of the stated lengths.

distributions are equal $\implies$ statistically close $\implies$ computationally indistinguishable

The converses may not be true. In particular, if one-way functions exist, then computational indistinguishability does not imply statistical closeness: $\{X_{2n}\}$ is the output distribution of a pseudorandom number generator $G : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$. $\{Y_{2n}\}$ is the output distribution of a true random number generator for $2n$ -bit strings. $\{X_{2n}\}$ contains at most 2^n different strings. $\{Y_{2n}\}$ contains 2^{2n} different strings. Therefore these distributions are statistically distinguishable. However, since G is a pseudorandom number generator, they are computationally indistinguishable.

3.2 Requiring the Simulator to Output the Coins of V

The output of the simulator consists of the messages passed back and forth between P and V' as well as the random bits used by V' . We will now show why it is necessary for the simulator to output the random bits of V' .

Consider the following protocol for graph isomorphism. This is clearly not a very good way to prove graph isomorphism, but it illustrates the need for the simulator to output the coins of V' . First, a description with words: The input is G_0 and G_1 . V randomly selects one of the two graphs, say G_0 , and sends to P a random permutation of that graph. P randomly selects one of the two graphs, say G_1 , and sends back the permutation which maps the graph it received from V to G_1 . If P and V happened to choose different graphs (in the above example V chose G_0 and P chose G_1), then V will be able to determine an isomorphism between G_0 and G_1 by composition of the permutations. Repeat everything k times, and if V is never able to determine an isomorphism between G_0 and G_1 , then V rejects. In table form:

$x = \{G_0 \sim G_1\}$		
P	communication	V
	$\xleftarrow{G'}$	Generate a random bit b and a random permutation Π_1 . Let $G' = \Pi_1(G_b)$.
Generate a random bit c . Choose a uniform isomorphism Π_2 such that $G_c = \Pi_2(G')$.	$\xrightarrow{\Pi_2}$	Record success if $G_{1-b} = \Pi_2(G')$.
		Run all k trials. Accept if at least one succeeds.

When $b = 0$ and $c = 1$, the isomorphism from G_0 to G_1 is $\Pi_2 \circ \Pi_1$. When $b = 1$ and $c = 0$, use its inverse.

This protocol is an interactive proof because:

- **Completeness:** If the graphs are isomorphic then V will determine an isomorphism whenever $b \neq c$. The probability that all k trials have $b = c$ is $1/2^k$, so the probability that P fails to convince V that the graphs are isomorphic is at most $1/2^k$.
- **Soundness:** If the graphs are not isomorphic then V will never accept, since it only accepts if it can determine an isomorphism between G_0 and G_1 .

The verifier learns an isomorphism between G_0 and G_1 . Thus an efficient simulator of its full view would also give an efficient way to find such an isomorphism. This is an obstruction under a hardness assumption, rather than an unconditional proof that no simulator exists.

However, if we only required the simulator to output the messages passed back and forth between P and V , then we could construct an appropriate simulator for the honest verifier V . Remember that to prove zero knowledge for a protocol, we would have to show how to simulate any verifier V' , not just V .

Code for (Pseudo)Simulator S_V

1. For $i = 1$ to k :
 - (a) Pick a random bit c .
 - (b) Pick a random permutation Π . Let $G' = \Pi^{-1}(G_c)$.
 - (c) Output the two messages shown below.

P	communication	V
	$\xleftarrow{G'}$	
	$\xrightarrow{\Pi}$	

These are generated messages, not an interaction with a verifier. For $x \in GI$, G' is uniformly distributed over all the graphs isomorphic to G_0 (or G_1 , since $G_0 \sim G_1$), and Π is a uniform map from G' to G_c , with equal probability for each value of c . This is exactly the distribution of the real messages.

Now consider also the random bits of V . If we have the above simulator output c as the bit supposedly used by V , the distribution is no longer correct. In the real conversation, the prover's bit c is independent of the verifier's bit b . In this attempted simulation they are always equal, and the returned permutation always maps G' back to the graph chosen by the verifier. In general this does not reproduce the real joint distribution. This justifies including the random bits of V in the view of what the verifier sees in the definition of a zero-knowledge interaction.

4 Proving a Protocol is Zero Knowledge

Let PV be protocol (P1) for graph isomorphism, defined earlier. It was proved previously that PV obeys Completeness and Soundness, so PV is an interactive proof. In this section we will prove that PV is zero knowledge. We will do this by constructing, for any verifier V' , a simulator whose distribution on $x \in GI$ is the same as that for $PV'(x)$. The simulator will depend on the notions of saving the state and restarting a Turing machine.

The verifier is a special kind of Turing machine. It consists of a finite state control, an input tape, output tape, work tape, random tape, input communication tape and output communication tape. The information on these tapes and the state of the control completely defines the state of the verifier. Thus, we can save the state of the verifier by saving this information. Say we save the state of the verifier at time t . We then put something on the verifier's input communication tape, let the verifier continue running and observe the behavior of the verifier. We can now restore the verifier to the state we saved at time t . If we now put the same thing on the verifier's input communication tape and let the verifier run again, we will observe exactly the same behavior we did before. The Turing machine has no way of remembering what we had it do before we restored its state. Its random tape is restored, not replaced by fresh coins.

4.1 Story-time

Jay Lentil, host of a popular late-night television show, convinces the great, world-famous magician Flippo to be on his show. To the amazement of all the viewers, Flippo proceeds to flip a normal coin and have it come up heads 100 times in a row. Not to be outdone, Dave Numberman, host of a competing show, tries to find a magician to match this incredible feat. Dave, however, doesn't find a suitable magician, so he disguises his assistant Paul in a magician's costume. Dave's show is pretaped, so in the studio, he has Paul flip a coin again and again until it has come up heads 100 times. Then the tape is edited to remove all the coin flips that came up tails. When the tape is shown that night, the viewers are amazed to see a man flipping a coin and having it come up heads 100 times in a row!

4.2 Construction of a Simulator for (P1)

The previous story showed how it is possible to run an experiment many times and pick out only the successful experiments. If the chance of success is high enough ($1/2$ in the story), then we can get the

required number of successful experiments quickly. We will use this for our simulator $S_{V'}$. We save the state of the verifier V' , run an experiment on V' , retain the results if they are successful, and otherwise restore the state of V' and run another experiment. After a successful experiment we proceed from the state reached by that experiment. We continue until we have the required number of successful experiments.

Code for Simulator $S_{V'}$

1. Pick a random tape R for V' .
2. For $i = 1$ to k :
 - (a) Record the state of V' : its finite state control, tape contents, and head positions.
 - (b) Repeat the following trial until $b = c$.

P (simulated by $S_{V'}$)	communication	V'
Pick a random bit c and a random permutation Π_i . Compute $H_i = \Pi_i(G_c)$.	$\xrightarrow{H_i}$	
	$\xleftarrow{b}$	Respond using the current state and random tape R .
If $b = c$, send Π_i , retain this conversation, and go to the next trial.	$\xrightarrow{\Pi_i \ (b = c)}$	
If $b \neq c$, reset V' to the recorded state and try again.		

Finally, output R and all retained messages. Failed attempts are not included.

Each experiment is successful half the time. In particular, it is successful when $b = c$. Since $G_0 \sim G_1$, the distribution of H_i is the same for $c = 0$ and $c = 1$. Thus c is independent of the verifier's challenge, even when the challenge depends on H_i . The expected number of trials is at most $2k$. Conditioning on $b = c$ does not change the distribution of H_i , and Π_i is a uniform isomorphism from G_b to H_i . This is precisely the real prover's response distribution. It follows, one trial at a time, that the full output distribution is identical to $[PV'(x)]$.

If V' stops or sends an invalid challenge, the real prover stops. For this trial only, the simulator treats the event internally as $b = 0$. It retains the stopped conversation when $c = 0$ and otherwise rewinds. No response is sent, and this internal bit is not recorded as a message. Stopping conversations therefore have the same retention probability $1/2$.

The simulator terminates with probability one and has expected polynomial running time. Hence this is already perfect zero knowledge. An exhaustive search is not needed. If we instead stop after k failed attempts in any one repetition, the probability of any simulation failure is at most $k/2^k$. This gives statistical zero knowledge with a polynomial bound on running time, subject to the sampling convention above.

4.3 Parallel Version of (P1)

Protocol (P1) runs a three-message protocol k times, so there are $3k$ total messages sent. Communication is expensive in reality, so we would like a way to minimize the number of rounds required by an interactive proof. This would save us the overhead on each separate message. One thing we could think of for (P1) is to do all k runs in parallel so that we have only three (albeit larger) messages sent. The Parallel (P1) protocol is:

$$x = \{G_0 \sim G_1\}$$

P	communication	V
For $i = 1$ to k : generate a random permutation Π_{1i} . Let $G'_i = \Pi_{1i}(G_0)$.	$\xrightarrow{G'_1, \dots, G'_k}$	
	$\xleftarrow{b_1, \dots, b_k}$	Generate random bits $b_1, \dots, b_k$.
For $i = 1$ to k : determine Π_{2i} such that $G'_i = \Pi_{2i}(G_{b_i})$.	$\xrightarrow{\Pi_{21}, \dots, \Pi_{2k}}$	Accept iff every map is a permutation and $G'_i = \Pi_{2i}(G_{b_i})$ for all i .

The above protocol is still an interactive proof for GI because:

- **Completeness:** If the graphs are isomorphic then P will be able to provide an isomorphism between G'_i and G_{b_i} for all i .
- **Soundness:** If the graphs are not isomorphic then V will only accept if some prover P' can guess right all k times. The probability that this happens and P' fools V is at most $1/2^k$.

Unfortunately, the simulator we created for (P1) will not work efficiently for Parallel (P1). A successful experiment would require it to guess all k bits $b_1, \dots, b_k$ correctly simultaneously. It can do this with probability only $1/2^k$. Its expected number of trials is therefore 2^k . Goldreich and Krawczyk showed that a three-message proof with negligible soundness error and black-box zero-knowledge simulation implies that the language is in BPP . The qualification “black-box” is essential. Below, we will see that with appropriate modifications to the protocol we can make a constant-round ZK proof for GI.

5 Number of Rounds for ZK

The topic of this section is the number of rounds of interaction needed for perfect and statistical ZK proofs. We will also consider the special subclass of interactive proofs (called Arthur–Merlin proofs) in which the verifier cannot keep secrets from the prover. We count each message as one round.

5.1 Arthur–Merlin Protocols

Arthur–Merlin proofs were introduced by Babai and Moran. The name is drawn from the Arthurian legend, where Merlin is the all-powerful prover and Arthur is the polynomial-time verifier. Arthur is restricted to generating public random coin tosses (i.e., which Merlin can see), as opposed to interactive proofs of [GMR], where the verifier can secretly generate the coin tosses. Moreover, the only messages Arthur is allowed to send to Merlin are the results of his coin tosses. Recall that in interactive proofs the verifier can make arbitrary polynomial computations in order to generate questions for the prover. In the setting where coins are public, there is no need to send anything but the coin flips, since whatever questions the verifier can compute based on his public coin flips, the prover can compute just as well.

Goldwasser and Sipser showed how to transform private-coin interactive proofs into public-coin proofs. The transformation need not preserve zero knowledge. Polynomially many public-coin rounds suffice for $IP = PSPACE$. This should not be confused with the usual class AM , which has a constant number of rounds. Under the existence of one-way functions, every language in IP has a computational zero-knowledge proof. Statistical zero knowledge is more restricted: $SZK \subseteq AM \cap coAM$.

5.2 Public/Private Coins and the Number of Rounds of Interaction

Let us revisit the proof for graph nonisomorphism in Section 2.3. In that protocol, the verifier generates a secret bit b and sends to the prover a random permutation $\Pi(G_b) = G'$ isomorphic to one of the two input graphs G_0 or G_1 . Then the prover sends back a bit indicating the graph to which G' is isomorphic. If the graphs G_0 and G_1 are isomorphic, the cheating prover will be caught with probability $1/2$. If the graphs are not isomorphic, the verifier will always be convinced.

Clearly, this protocol is not public coin. The verifier cannot publicly generate the coin flips in this protocol: revealing b would reveal the answer. The protocol is zero knowledge for the honest verifier,

since a simulator can choose b and Π itself and write down the answer b . But this does not establish zero knowledge against arbitrary verifiers. Suppose the verifier had a third graph G_{new} known to be isomorphic to either G_0 or G_1 , without knowing which one. The cheating verifier could substitute G_{new} for G' and have the prover identify which graph it is isomorphic to. The honest-verifier simulator does not explain how to produce this answer. Without a hardness assumption, this is not an unconditional impossibility proof.

Could we construct a ZK proof for GNI which is also public coin? The answer depends on the number of rounds. Goldreich and Krawczyk showed that a constant-round public-coin proof with negligible soundness error and black-box zero-knowledge simulation for a language L implies $L \in BPP$. However, if we allow private coins, we can design constant-round perfect ZK proofs for GI and GNI. We first look at the graph-isomorphism protocol due to Bellare, Micali and Ostrovsky [BMO90].

5.3 Private Coins: Constant-Round Protocol for Graph Isomorphism

Let's look again at the graph-isomorphism protocol we devised earlier. In that protocol, the prover sends a random graph C isomorphic to the two input graphs G_0 and G_1 , obtained by choosing one of them at random and randomly permuting it. Then the verifier sends a random bit b , and the prover has to show the isomorphism between C and G_b .

If the graphs are indeed isomorphic, the verifier will always be convinced. If they are not, the cheating prover will be caught with probability $1/2$. We want to amplify the probability of catching a cheating prover. Repeating the atomic protocol k times sequentially will amplify the probability to $1 - 2^{-k}$ and will also preserve the ZK property. But this requires repeating the atomic protocol sequentially, so the number of messages rises to $3k$.

Can we squeeze all k repetitions into fewer steps? We considered sending all k graphs isomorphic to the input graphs at once, then all k query bits, then all the answers. This protocol is still an interactive proof, but the ZK property cannot be established by the preceding simulator.

The problem is that when the simulator sends the k graphs $H_1, \dots, H_k$ to a verifier, the verifier may send in return query bits that depend on those graphs. This means that if the simulator tries to reset the verifier and send k new graphs generated from G_0 and G_1 according to the query bits, the verifier might ask different queries. Depending on luck is also a bad strategy here. While the verifier has no way to know from which of the input graphs each H_i was generated, the chance that all the query bits will match the generation pattern of the H_i 's is only 2^{-k} . So this simulator takes exponential expected time to generate a valid conversation. The verifier might choose the query bits according to a complicated function of the H_i 's.

What should we do? Intuitively, we would like to modify the protocol so that the verifier will have to commit to its query bits before seeing the H_i 's. Of course the commitment should not reveal the bits to the prover in any way. Otherwise it might generate the graphs H_i accordingly, and the protocol would no longer be an interactive proof. Having such a bit commitment mechanism, the idea is that a simulator will be able to get the commitments to the bits, then send some $H_1, \dots, H_k$. The verifier then decommits, or reveals its bits. Now the simulator rewinds the verifier to the state it was in just before receiving the H_i 's, and sends another sequence of k graphs generated according to those bits.

We face, however, a major problem. Computational encryption can hide information from a polynomial-time verifier. But our prover has unbounded computational power, so the verifier cannot hide anything merely by computational encryption. The solution is to modify our requirements from the bit commitment protocol. We will devise a mechanism that ensures that the prover has no way at all to know the query bits, since this is essential for maintaining the IP properties. It is also possible that the verifier may cheat, that is, reveal a different query than the one it committed to. This has no relevance to the IP properties, but it might affect the ZK property. However, two valid openings to different bits will provide the simulator with an isomorphism between the two commitment graphs.

In the following table, an arrow labeled by an isomorphism statement means that the actual vertex permutation is sent, not just the statement that an isomorphism exists. All the maps in the last row are sent together.

Constant-round GI: five messages

P	communication	V
Generate two random graphs A_0, A_1 by independently permuting G_0 twice.	$\xrightarrow{A_0, A_1}$	
	$\xleftarrow{\Pi_1(A_{b_1}), \dots, \Pi_k(A_{b_k})}$	Generate random bits $b_1, \dots, b_k$ and independent random permutations $\Pi_1, \dots, \Pi_k$.
Generate random graphs $H_1, \dots, H_k$ by independently permuting G_0 .	$\xrightarrow{H_1, \dots, H_k}$	
Check that all (b_i, Π_i) are valid openings of the graphs received. If this is not the case, stop the conversation.	$\xleftarrow{(b_1, \Pi_1), \dots, (b_k, \Pi_k)}$	Send the query bits with their opening permutations.
Show $G_0 \sim A_0 \sim A_1$ and show $H_i \sim G_{b_i}$ for every i .	$\xrightarrow{G_0 \sim A_0 \sim A_1, H_1 \sim G_{b_1}, \dots, H_k \sim G_{b_k}}$	Check every isomorphism. Accept only if all checks pass.

The proof that $G_0 \sim A_0 \sim A_1$ may be supplied by the two original permutations from G_0 to A_0 and A_1 . For the H_i 's, use the permutations that generated them, composed with a fixed isomorphism from G_1 to G_0 when $b_i = 1$, just as in (P1).

It is easy to see that the protocol is indeed an IP protocol. Completeness is perfect. For soundness, if $A_0 \not\sim A_1$, then the final check fails. Otherwise, the random permutations of A_0 and A_1 have exactly the same distribution. Thus the commitment graphs reveal no information about $b_1, \dots, b_k$, even to an unbounded prover. If $G_0 \not\sim G_1$, then each graph H_i can answer at most one of the two possible queries. The probability that every query can be answered is at most 2^{-k} .

The simulation idea

To prove that the protocol is ZK we have to show a simulator for any verifier. The simulation uses two modes. In the *honest mode*, the simulator generates A_0 and A_1 from G_0 , sends them to the verifier, then gets the k permuted copies of them (the commitments), and sends k random graphs generated from G_0 . The verifier then decommits and reveals the bits. The simulator rewinds the verifier to the state just before the k random graphs were sent, and now sends k permuted graphs generated from G_0 or G_1 according to the query bits. If the verifier decommits to the same bits, the simulator can answer all the queries and show that A_0 and A_1 are copies of G_0 .

If, however, the verifier sends different bits, the simulator cannot complete this attempt in the same way. Note that it cannot simply output “the verifier cheated,” since this is not something the prover could find out in a real conversation. This motivates the second mode, the *cheating mode*. Now the simulation starts by generating A_0 from G_0 and A_1 from G_1 . The verifier cannot distinguish between these modes, since on a true GI input it always sees two independent random isomorphic graphs. The simulation proceeds as before: the verifier is rewound and given new H_i 's.

If the verifier decommits to the same bits, the simulator is again out of luck, since it cannot demonstrate the isomorphism between A_1 and G_0 . But if the verifier changes a bit, consider an index i where it does so. In the first try it demonstrated that its i th commitment graph is isomorphic to A_0 , say. In the second try it demonstrated that the same graph is isomorphic to A_1 . By composing one map with the inverse of the other, the simulator obtains an isomorphism between A_0 and A_1 , and therefore between G_0 and G_1 . Both openings must be valid for this extraction to work. An invalid opening or an early stop must instead be handled as an aborting conversation.

The protocol which appears in [BMO90] is a perfect ZK protocol. The argument above explains the two modes, but is not by itself a complete simulation proof. Simply alternating modes and retaining successful runs can bias the distribution. One must preserve the probabilities of the two cases and

of aborting conversations, and generate the final maps with the real prover's distribution. The full simulator and its expected polynomial-time analysis appear in [BMO90, Sections 5–6].

Theorem (Bellare, Micali, Ostrovsky). *Graph isomorphism has a five-message perfect zero-knowledge interactive proof with perfect completeness and soundness error at most 2^{-k} . The simulator runs in expected polynomial time.*

6 Perfect Zero-Knowledge Graph Nonisomorphism

In this section we construct a ZK protocol for GNI, due to [GMW].¹ Let $L = \{(G_0, G_1) \mid G_0 \not\sim G_1\}$. The prover's task is to convince the verifier that two graphs G_0 and G_1 are not isomorphic.

Recall the basic GNI protocol: the verifier sends a randomly permuted copy G' of G_b , and the prover returns b . If $G_0 \not\sim G_1$, the prover can always tell b . On the other hand, if $G_0 \sim G_1$, the prover can send the correct bit at most half the time.

The difficulty with zero knowledge is that a verifier may send a graph C such that $G_0 \sim C$ or $G_1 \sim C$, without knowing which is the case. If the prover answers faithfully, the verifier learns which graph is isomorphic to C . The honest-verifier simulation does not explain how to produce that answer. To design a ZK proof, let us first consider a modification of the basic protocol.

6.1 The Paired Protocol

Here is another interactive protocol for graph nonisomorphism. P wants to prove to V that $G_0 \not\sim G_1$.

P	communication	V
	$\xleftarrow{C_0, C_1}$	Flip a coin $b \in \{0, 1\}$. Using independent random permutations, generate $C_0 \sim G_b$ and $C_1 \sim G_{1-b}$.
Find bit b .	$\xrightarrow{b}$	Accept if the returned bit is correct.

If no unique bit satisfies both isomorphism conditions, the prover sends a failure symbol, which is rejected.

This is an interactive protocol:

- If $G_0 \not\sim G_1$, then $C_0 \not\sim C_1$, so P can always distinguish between them and thus determine the bit.
- If $G_0 \sim G_1$, then the pair (C_0, C_1) has the same distribution for both choices of b . Thus any prover has at most a $1/2$ chance of fooling V . This can be reduced to $1/2^k$ by repeating the process k times.

However, the protocol is not clearly zero knowledge: we run into difficulties constructing a simulator for it. The basic problem is that V may not know the bit b before the prover tells it what it is, so that it actually gains information. To overcome this, we force V to start by proving to P that it knows b , so that it cannot have gained anything from P 's answer.

6.2 The Expanded Protocol

Here is the expanded protocol. In this table, $D_1, \dots, D_{2k}$ are generated by independent random permutations. Within each pair, a fresh fair coin decides which graph is a copy of C_0 and which is a copy of C_1 . This random ordering is necessary. The graphs C_0, C_1 and all the D graphs are sent together as the first message.

¹Scribe notes by Shuzo Takahashi, October 18, 1994, and Sanjoy Dasgupta, October 27, 1994.

Expanded GNI protocol

P	communication	V
	$\xleftarrow{C_0, C_1} D_1, \dots, D_{2k}$	Flip a coin b . Generate $C_0 \sim G_b$ and $C_1 \sim G_{1-b}$. Generate $D_1, \dots, D_{2k}$ with $\{D_{2i-1}, D_{2i}\} \sim \{C_0, C_1\}$, using the independent random permutations and pair orderings described above.
Check graph sizes. Flip coins $q_1, \dots, q_k$.	$\xrightarrow{q_1, \dots, q_k}$	Find k pair isomorphisms $\pi_1, \dots, \pi_k$. If $q_i = 0$, use $\pi_i : \{D_{2i-1}, D_{2i}\} \rightarrow \{G_0, G_1\}$. If $q_i = 1$, use $\pi_i : \{D_{2i-1}, D_{2i}\} \rightarrow \{C_0, C_1\}$.
Check all the π_i . Stop if any check fails.	$\xleftarrow{\pi_1, \dots, \pi_k}$	Send the pair isomorphisms.
Find the unique bit b such that $C_0 \sim G_b$ and $C_1 \sim G_{1-b}$.	$\xrightarrow{b}$	Accept if the returned bit is correct.

As before, if no unique bit exists, the prover sends a failure symbol. All indicated checks occur before it sends its answer.

Notation:

$$\{A, B\} \sim \{C, D\} \text{ means } (A \sim C \wedge B \sim D) \text{ or } (A \sim D \wedge B \sim C)$$

Here braces mean an unordered pair, retaining multiplicities. $\pi : \{A, B\} \rightarrow \{C, D\}$ means that π specifies the pairing and the two relevant vertex permutations.

To get an idea of what is going on in the “proof of knowledge” phase, consider the pair (D_1, D_2) . P will ask V to demonstrate either a mapping $\{D_1, D_2\} \rightarrow \{G_0, G_1\}$ or a mapping $\{D_1, D_2\} \rightarrow \{C_0, C_1\}$. If we obtain valid answers to both questions, we obtain a mapping $\{G_0, G_1\} \rightarrow \{C_0, C_1\}$ by inverting the first map and composing it with the second. On a true GNI input, this determines b . More generally, any two distinct challenge strings with valid responses differ at some position i , and that position gives the same extraction. Two different questions alone do not suffice: both responses must be valid.

6.3 Completeness and Soundness

If $G_0 \not\sim G_1$, the honest verifier knows all the indicated mappings, and the prover can always determine b . Completeness is perfect.

If $G_0 \sim G_1$, the initial graphs have the same distribution for both values of b . The opening permutations do not change this. To see why, fix the graphs already sent. A uniform random permutation, conditioned on its image graph, is a uniform isomorphism to that graph. For $q_i = 1$, the verifier opens a fresh random ordering of copies of C_0, C_1 . For $q_i = 0$, it composes those maps with its maps from the C pair to the G pair. Composition is a bijection on the possible maps. The random ordering makes the pairing uniform in either case. Consequently the full response to any chosen challenge string has the same distribution for $b = 0$ and $b = 1$. Even a cheating prover cannot guess b with probability greater than $1/2$.

6.4 A Statistical ZK Simulator

Here is a statistical ZK simulator for this protocol. Fix a random tape for V' once. A cheating verifier need not answer a second random challenge correctly, so extraction may require more trials.

1. Get (C_0, C_1) and $(D_1, D_2), \dots, (D_{2k-1}, D_{2k})$ from V' . If the graphs are malformed or V' stops, produce the same stopped view as the real prover.
2. Record V' 's state.
3. Send random bits $q_1, \dots, q_k$ and get responses $\pi_1, \dots, \pi_k$. Save this conversation and the state after the response. If a response is invalid, output this aborting conversation and stop.

4. Reset V' to its state in step 2. Send fresh random bits $q'_1, \dots, q'_k$ and get responses $\pi'_1, \dots, \pi'_k$. If all responses are valid and $q_i \neq q'_i$ for some i , use π_i and π'_i to determine b . Otherwise repeat this step, for at most 2^k attempts.
5. On extraction success, restore the state after the original response in step 3, send b , and output that original conversation and the verifier's coins. The extraction attempts are not part of the output. If all 2^k attempts fail, output a simulation-failure symbol.

The interaction in steps 3–5 has the following form:

P (simulated by $S_{V'}$)	communication	V'
Save the original challenge and the resulting verifier state.	$\xrightarrow{q_1, \dots, q_k}$	
	$\xleftarrow{\pi_1, \dots, \pi_k}$	
Rewind to the state in step 2 for each extraction attempt.	$\xrightarrow{q'_1, \dots, q'_k}$	
Check validity; compare the challenge strings; extract when possible.	$\xleftarrow{\pi'_1, \dots, \pi'_k}$	
Restore the saved state after the <i>original</i> response.	$\xrightarrow{b}$	Continue the original conversation.

For the running-time calculation, fix the verifier's tape and its state in step 2. Let a be the number of the 2^k challenge strings that receive valid responses. The simulator enters the extraction step with probability $a/2^k$. If $a \geq 2$, a new trial finds a different valid challenge with probability $(a - 1)/2^k$. Including the probability of entering extraction, the expected number of trials is at most

$$\frac{a}{2^k} \cdot \frac{2^k}{a - 1} = \frac{a}{a - 1} \leq 2$$

If $a = 1$, the simulator makes 2^k trials with probability $1/2^k$, for expected cost one trial. If $a = 0$, it makes no extraction attempts. Thus the expected running time is polynomial.

For $a \geq 1$, the probability of simulation failure is at most

$$\frac{a}{2^k} \left(1 - \frac{a - 1}{2^k}\right)^{2^k} \leq \frac{ae^{-(a-1)}}{2^k} \leq \frac{1}{2^k}$$

Couple the simulation with a real execution using the same verifier coins and original challenge. Their views can differ only on simulation failure: otherwise the simulator preserves the original challenge and response and supplies precisely the bit the real prover would return. Therefore the statistical distance is at most 2^{-k} . This does not claim equality of the distributions conditioned on simulation success. The simulation runs in expected polynomial time, not strict polynomial time.

6.5 Constant-Round ZK for GNI

To reduce the soundness error without increasing the number of rounds, run k copies of the expanded protocol in parallel. Each copy uses its own bit b , its own C_0, C_1 , and its own k pairs of D graphs. The verifier uses independent randomness in all copies. Corresponding messages are sent together. The prover checks *all* openings before returning *any* answers.

Constant-round GNI: four messages

P	communication	V
	$\xleftarrow{\text{all } C \text{ pairs and } D \text{ graphs}}$	Prepare k independent copies of the expanded protocol.
For each copy, choose its k independent challenge bits.	$\xrightarrow{\text{all challenge bits}}$	
Check every opening in every copy. Stop if any check fails.	$\xleftarrow{\text{all pair isomorphisms}}$	Answer every challenge as in the expanded protocol.
Determine the bit for each copy.	$\xrightarrow{b_1, \dots, b_k}$	Accept only if every returned bit is correct.

Completeness remains perfect. On a false input, the full information given to the prover is independent of the k hidden bits, even when its challenges depend on all the first messages. Thus the probability of guessing all k bits correctly is at most 2^{-k} .

For simulation, first choose all challenge bits and save the original conversation. If any original opening is invalid, output the real aborting view. Otherwise extract one copy at a time. To extract a copy, rewind to the state before the challenges, keep the original challenge bits for all other copies fixed, and resample only this copy's k bits. Check the responses for this copy and use a different valid response to obtain its bit. Make at most 2^k extraction attempts for each copy. If any copy's extraction fails, output a simulation-failure symbol. After all bits are found, restore the state after the original responses and send all the bits together.

The preceding calculation applies to each copy. For fixed challenges in the other copies, let a count the challenge strings receiving valid responses in the copy being extracted. The chance that *all* original responses are valid is at most $a/2^k$, which is all the calculation needs. Therefore each copy contributes at most two expected extraction trials and at most 2^{-k} failure probability. Summing over the copies gives expected polynomial time and statistical distance at most $k/2^k$. This establishes statistical zero knowledge for the four-message protocol. It is a direct simulation of this protocol, not a general assertion that parallel repetition preserves zero knowledge.

6.6 Perfect Zero Knowledge

To make the simulator perfect ZK, we can augment it with a low-probability exponential-time search. If the capped extraction for a copy fails, search exhaustively for the mappings between its C pair and (G_0, G_1) . Return the unique bit if one exists, or the same failure symbol that the real prover would return otherwise. Restore the state after the original response and continue the original conversation.

Choose k polynomially large in the input length so that the time for this exhaustive search is at most 2^k . Such a choice is possible: testing all vertex permutations takes at most a factorial times a polynomial in the number of vertices. The search is needed with probability at most 2^{-k} for each copy, so its contribution to the expected running time is at most one per copy. The simulator always returns the real prover's answer and retains the original conversation. Its output distribution is therefore exactly the real distribution.

Claim. *The four-message GNI protocol has perfect completeness, soundness error at most 2^{-k} , and perfect zero knowledge with expected polynomial-time simulation, for the parameter choice above. Without the exhaustive-search fallback, the displayed simulator has statistical error at most $k/2^k$.*

7 Another Example: Proving an OR of GI

Let

$$L = \{((G_0, G_1), (C_0, C_1)) \mid G_0 \sim G_1 \text{ or } C_0 \sim C_1\}$$

In the following protocol, the prover needs to convince the verifier that at least one of two pairs of graphs is isomorphic.

P	communication	V
Randomly generate bits b_1, b_2 and uniform random copies G', C' such that $G' \sim G_{b_1}$ and $C' \sim C_{b_2}$.	$\xrightarrow{G', C'}$	
	$\xleftarrow{b}$	Randomly generate a bit b .
Choose b'_1, b'_2 such that $b = b'_1 \oplus b'_2$ and supply both isomorphisms $G_{b'_1} \rightarrow G'$ and $C_{b'_2} \rightarrow C'$.	$\xrightarrow{\text{both isomorphisms } b'_1, b'_2}$	Check the XOR condition and both isomorphisms.

Repeat the above k times sequentially. The prover consistently adjusts the bit for a pair whose isomorphism it knows and leaves the other bit unchanged.

If $x \notin L$, that is, $G_0 \not\sim G_1$ and $C_0 \not\sim C_1$, then for fixed G', C' at most one XOR challenge can be answered. So the verifier rejects with probability at least $1/2$. On the other hand, if $x \in L$, say $G_0 \sim G_1$, then the prover can change b_1 to b'_1 if necessary and take $b'_2 = b_2$, so that $b = b'_1 \oplus b'_2$. The prover can always convince the verifier.

The following is a simulator. Record the state of the verifier. Pick b_1, b_2 at random and generate G', C' as in the protocol. Send G' and C' to the verifier. If $b = b_1 \oplus b_2$, supply b_1, b_2 and the two known isomorphisms. Otherwise reset the verifier to the recorded state and repeat. Here b is the verifier's actual response, not necessarily an independently random bit.

On a true input, at least one of b_1, b_2 remains uniform even after G', C' are fixed. The XOR is therefore uniform, and each trial succeeds with probability $1/2$. The retained messages and maps have exactly the real distribution. The simulator terminates with probability one in expected polynomial time, so the protocol is perfect zero knowledge. Invalid challenges and early stopping are handled as for (P1). Truncating after k attempts per repetition gives statistical error at most $k/2^k$.

References

- [GMR] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing* 18(1):186–208, 1989. Preliminary version, STOC 1985.
- [GMW] O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM* 38(3):690–728, 1991. Paper.
- [BMO90] M. Bellare, S. Micali and R. Ostrovsky. Perfect zero-knowledge in constant rounds. *STOC 1990*, pp. 482–493. Full version.
- [GK] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing* 25(1):169–192, 1996. Paper.