In this lecture we describe the main ingredient that will allow us to turn any two-party protocol that is secure for honest-but-curious parties into one that is secure even when the parties are malicious: zero-knowledge proofs.

As we described in the last lecture, the idea is to augment each message that Bob sends to Alice (and vice versa) by some additional interaction whose purpose it is to certify that Bob acted honestly. One way for Bob to prove he is honest is by revealing his private input and randomness. In that case, Alice can check that Bob is behaving the way he is supposed to. However, Bob's privacy is then compromised.

Zero-knowledge proofs are a technology that allow Bob to convince Alice of the validity of some statement (in this case, the statement that his message is consistent with his private input and randomness) without revealing any private information. In fact, after the zero-knowledge interaction Alice will find no information about Bob beyond the fact that the statement in question is true. Alice will not be able to "extract" any additional information about Bob even if she behaves in a malicious way.

Instead of trying to provide zero-knowledge proofs for a specific protocol that we may be interested in, it is more instructive (and conceptually simpler) to develop a theory of zero-knowledge proofs that applies to any statement of interest. To describe this theory, we will need a bit of computational complexity. Before we do that let us start with some examples that illustrate some of the ideas behind zero-knowledge.

1 A real-world zero-knowledge interaction

Before we go into zero-knowledge for two-party computations, let me give a "real-world" example of a zero-knowledge interaction. Coca Cola and Pepsi are two popular drinks of a similar taste. Some people claim that Coca Cola is much better and they would never touch Pepsi, while others say the opposite. Yet others cannot tell the difference and wonder what the whole fuss is about.

Now imagine that Bob comes to Alice and tells her that Coke tastes much better than Pepsi. Alice, who cannot tell the difference between the two, is quite incredulous and decides to put Bob to the test. To do so, she buys some Coke and Pepsi from the market as well as ten plastic cups. She pours Coke in the first five cups, Pepsi in the other five, and permutes the cups in the random order, remembering what she put in each cup. Bob takes a sip from all the cups and tells Alice which ones have Coke and which ones have Pepsi in them.

If Bob is the connoisseur he claims to be, he can correctly tell the contents of all the cups. If, on the other hand, his claim that he can distinguish between the two drinks is false, then all $\binom{10}{5}$ orderings of the cups look alike to him, and no matter what strategy he uses to make his claims, the chance that he gets all the answers correct is at most $1/\binom{10}{5} < 4\%$. So if Bob managed to pass the test, Alice can have good confidence that Bob can tell the difference between the two drinks.

This is a (real-world) example of an *interactive proof*: After receiving a claim from Bob and engaging in an interaction with him, Alice can say with good confidence that Bob's claim is correct.

However, after engaging in this interaction, Alice obtained some additional information beyond the fact that Bob can tell the difference between Coke and Pepsi. Suppose that the bottles from which Alice was pouring drinks were unlabeled, so she doesn't know which has Coke and which has Pepsi. After the interaction, she is still convinced that Bob can tell the difference (because he gives consistent answers). However, assuming the Bob is honest, she also gains some additional information, for example what is the drink in the first glass.

But now suppose they change the rules of the game and play like this: Before Bob gives his answers, he chooses random names for the two drinks, say "drink A" and "drink B". With probability $1/2$ he labels Coke as "drink A" and Pepsi as "drink B", and with probability $1/2$ he labels Pepsi as "drink A" and Coke as "drink B". Then he answers the questions not by Coke and Pepsi, but by drink A and drink B instead.

Now here is what an interaction between Alice and Bob will look like. Alice has two bottles, bottle 1 and bottle 2, one of which contains Coke and the other one Pepsi (but she is not sure which). She wants to know Bob can tell the difference between the two drinks. She pours five glasses from the first bottle and five from the second bottle, permutes the glasses randomly and presents them to Bob:

$$1 \quad 1 \quad 2 \quad 1 \quad 2 \quad 2 \quad 2 \quad 1 \quad 2 \quad 1.$$

If Bob is dishonest, then all of the glasses look the same to him, so when asked what is in each glass, he will give some random jumble of As and Bs, which are unlikely to correspond to this pattern of 1s and 2s. If Bob can tell the difference between Coke and Pepsi, then he will know which drink is in each glass. After choosing a random labeling for Coke and Pepsi, he comes up with the following answer:

$$\begin{array}{cccccccccc} A & A & B & A & B & B & B & A & B & A \quad \text{with probability } 1/2, \\ B & B & A & B & A & A & A & B & A & B \quad \text{with probability } 1/2. \end{array} \tag{1}$$

Either way, Alice can be quite certain Bob can tell the difference. However, she has learned no additional information from the interaction: She can simulate the distibution (1) all by herself!

2 Graph isomorphism

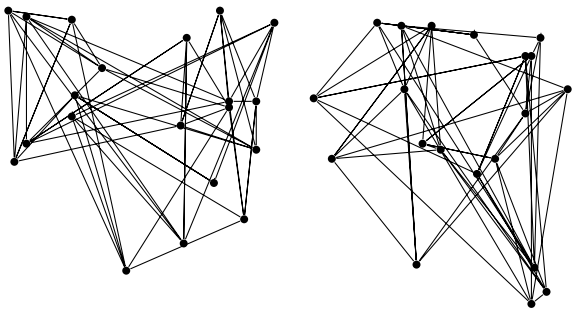
Let us now give a digital analogue of this real-world example. The example is a bit contrived, but it will help us illustrate some definitional issues regarding zero-knowledge.

We say two undirected graphs G and H on n vertices are *isomorphic* if there is a 1-1 correspondence between the vertices that preserves the edges. More precisely, there is a 1-1 map ϕ from the vertices of G to the vertices of H so that (u, v) is an edge in G if and only if $(\phi(u), \phi(v))$ is an edge in H. For example, the following two graphs G and H are isomorphic:

The isomorphism that maps G to H is given by

x:	1	2	3	4	5
$\phi(x)$:	1	3	5	2	4.

In this example, it is easy to see that the two graphs are isomorphic. But it may not be so easy when the graphs are large:



Now suppose Alice and Bob are both given two graphs G and H and Bob claims that he knows an isomorphism ϕ from G to H. How can he convince Alice this is the case? One thing he can do is send Alice the values $\phi(x)$ for every x, at which point Alice can check that ϕ is an isomorphism: First, she checks that ϕ is a permutation over the set $\{1, \ldots, n\}$ where n is the number of vertices of G and H and then she verifies that for every pair of vertices (u, v), (u, v) is an edge in G if and only if $(\phi(u), \phi(v))$ is an edge in H.

However, after seeing Bob's proof, in addition to learning that G and H are isomorphic, Alice gains some additional knowledge – the isomorphism between G and H. Is there a way for Bob to convince Alice that G and H are isomorphic without revealing the isomorphism?

We will now see how to do so. In the description of the protocol we will use the following piece of notation: Let G be a graph and ϕ be a permutation on the vertices of G. We will write $\phi(G) = I$ if I is the graph obtained by applying the permutation ϕ to the vertices of G and permuting the edges accordingly (i.e., $(\phi(u), \phi(v))$ is an edge of I if and only if (u, v) is an edge of G.)

Graph Isomorphism Protocol: Alice and Bob are given a pair of graphs (G, H) on n vertices. In addition, Bob has a permutation (isomorphism) ϕ such that $\phi(G) = H$.

- B: Choose a random permutation ρ on the set of vertices of G. Send the graph $\rho(G)$ to Alice.
- A: Upon receiving the graph I, choose a random bit $b \sim \{0, 1\}$. If $b = 0$, send the message `G` to Bob. If $b = 1$, send the message `H` to Bob.
- B: If you receive the message `G`, send the permutation ρ to Alice. If you receive `H`, send the composed permutation $\rho \circ \phi$ to Alice. (This is the permutation given by $(\rho \circ \phi)(u) = \rho(\phi(u))$.)
- A: Upon receiving the permutation π: If you sent `G`, accept iff $\pi(G) = I$. If you sent `H`, accept iff $\pi(H) = I$.

You can think of the message `G` as a request by Alice to see an isomorphism between G and I, and the message `H` as a request for an isomorphism between H and I.

If the graph G and H are isomorphic and Alice and Bob both follow the protocol, Alice will clearly always accept: The graph I she sees is isomorphic to both G and H, and Bob can produce the required isomorphism in either case.

What happens if G and H are not isomorphic? Then ϕ cannot be an isomorphism between G and H, and the protocol we described is ill-defined. However, a malicious Bob can still attempt to trick Alice into believing that G and H are isomorphic. What are his chances of success? In the first round, Bob sends some graph I to Alice, which may or may not be isomorphic to G or H. The key observation is that if G and H are not isomorphic, then I cannot be isomorphic to both of them at the same time. If I is not isomorphic to G, then with probability $1/2$ in round two Alice will ask to see an isomorphism between G and I, which Bob will be unable to produce. If I is not isomorphic to H, Bob similarly fails with probability $1/2$. So no matter what strategy Bob uses, he will fail to convince Alice with probability at least $1/2$.

This is quite similar to the Coke versus Pepsi test, except that the probability of failure is lower – $1/2$ as opposed to $1 - 1/\binom{10}{5}$. By repeating the protocol k times independently, this probability can be made as high as $1/2^k$.

Now suppose again that G and H are isomorphic and Bob is honest. What has Alice learned after interacting with Bob? First she sees a graph I and then she sees some permutation π. If she chooses $b = 0$, then Bob sends her the permutation $\pi = \rho$ such that $\pi(G) = I$; but Alice could have easily come up with this view by herself by first choosing π at random, and then setting $I = \pi(G)$! If she chooses $b = 1$, then Bob sends her the permutation $\pi = \rho \circ \phi$ for which $\pi(H) = I$. But no matter what ϕ is, if ρ is random, then π will also be random, and so again Alice can simulate her view by first choosing a random permutation π and then choosing I such that $\pi(H) = I$. In either case, Alice has gained no information by running the protocol (beyond the knowledge that G and H are isomorphic)!

Notice that in this discussion, we never assumed that Alice had to behave honestly. The place in the protocol where Alice can possibly cheat is by choosing her bit b in some way that is not random; perhaps after seeing ρ in the first round, she can try to bias b in a way that allows her to gain some extra information. We just argued that this is not possible: Even if Alice is dishonest, whatever she has observed after the interaction, she would be able to simulate without engaging in the interaction at all.

3 Zero-knowledge proofs

Let us summarize the properties of the graph isomorphism protocol we just analyzed:

- If Bob has an isomorphism ϕ between G and H and both Alice and Bob behave honestly, Alice accepts with probability one.
- If G and H are not isomorphic, then no matter what Bob does, if Alice is honest she accepts with probability at most $1/2$.[1]
- If Bob has an isomorphism ϕ between G and H and he is honest, then no matter what Alice does, she gains no knowledge from the interaction (beyond the fact that G and H are isomorphic). Specifically, Alice can run a simulator that produces a distribution on views by Alice that is identical to her distribution on views in the real interaction.

[1] We do not make any requirement when G and H *are* isomorphic, but Bob is not given the isomorphism ϕ.

This is a bit different from the setting of secure two-party computation. In secure two-party computation, Alice and Bob have symmetric roles: Each gets a private input, and they want to compute a joint functionality without revealing their inputs. In zero-knowledge, the role of Bob is to *convince* Alice of the truth of some statement – that her two graphs are isomorphic. He could easily do so by providing the solution – the isomorphism. However, he wants to do it in a way that preserves the privacy of his solution.

Now let's consider under what circumstances it would make sense for Alice and Bob to behave maliciously. If ϕ is an isomorphism from G to H, he has no incentive of cheating, as Alice cannot find out anything from the interaction. On the other hand, if ϕ is not an isomorphism from G to H, Bob is in some sense forced to cheat, so we do not impose any restrictions on the knowledge gained by Alice. (Bob can always prevent Alice from gaining any knowledge by sending dummy messages.)

To define zero-knowledge proofs we first need to define what is a proof. Here we will view proofs from a verification perspective: Suppose you are grading homeworks and you need to check that all the solutions are correct. You don't really care about how the students came up with their solutions (plagiarism issues aside).

A *proof relation* R is a relation over pairs (x, y) called *the statement* and *the proof*. We view the statement $(x, y) \in R$ as saying that y is a correct proof of x. In the graph isomorphism example, x is the pair of graphs (G, H), y is the isomorphism ϕ, and $((G, H), \phi) \in R$ if and only if $\phi(G) = H$.

Definition 1. A (s, ε) *zero-knowledge proof system* with *simulation overhead* $oh(\cdot)$ for proof relation R over pairs (x, y) is a pair of interactive algorithms (A, B) where

- **Completeness:** If $(x, y) \in R$, then $\Pr_{(A(x),B(x,y))}[A \text{ accepts}] = 1$.
- **Soundness:** If $(x, y^*) \notin R$ for every y^*, then for every B^*,

$$\Pr_{(A(x),B^*(x))}[A \text{ accepts}] \leq 1/2.$$

- **Zero-knowledge:** If $(x, y) \in T$, then for every algorithm A^*, there exists an algorithm S^* such that $\text{view}_{A^*}(x, y)$ and $S^*(x)$ are (s, ε) computationally indistinguishable. Moreover,

$$\text{running time of } S^*(x) \leq oh(\text{running time of } A^*(x))$$

(where the running time is measured with respect to the interaction $(A, B)(x, (x, y))$.)

In completeness and soundness, the probability is taken over the internal randomness of the parties in the protocol. Let us now explain the zero-knowledge condition. Notice the soundness condition assumes not only that Bob's candidate solution y is invalid for x, but that no such solution y^* exists at all.

Informally, this condition says that every view by Alice – even if she is malicious – in the real interaction can be simulated. But how much time should the simulation take? We cannot give a specific bound, because the time it takes (a malicious) Alice to perform the simulation may depend on the time she is willing to invest to break privacy. It seems reasonable to require that the time invested by Alice for simulation related, and not too much larger than, the time it takes Alice to run the protocol. We formalize this by giving a *simulation overhead function* $t(\cdot)$ that relates the running time of the simulator to the running time of Alice in the actual protocol. Intuitively, the lower the simulation overhead is, the more secure the protocol should be.

Simulation overhead for graph isomorphism Let us compute the simulation overhead in the graph isomorphism protocol. When computing this quantity we have to be careful and consider as part of Alice's view not only the messages she received from Bob, but also the ones she sent to him.

Alice's view in the graph isomorphism protocol is (I, b, π, d), where ϕ is the permutation she receives in the first message, b is her response (`G` or `H`), I is the graph she receives in the third round, and d is her decision to accept or reject. Let A^* be any interactive algorithm that plays Alice's part in a possibly malicious way.

An elegant way to simulate $\text{view}_{A^*}(x, y)$ is like this: First, S^* guesses the value $b^* \sim \{0, 1\}$ at random and chooses a random permutation π. If $b^* = 0$, it sets $I = \pi(G)$; if $b^* = 1$, it sets $I = \pi(H)$. It then runs $A^*(x)$, simulating Bob's first message as I. If $b^* = b$, it simulates Bob's third message as π, then runs A^* to compute the decision d and outputs (I, b, π, d). If $b^* \neq b$, it restarts the whole simulation from scratch.

Conditioned on $S^*(x)$ producing an output, the output is then identically distributed to $\text{view}_{A^*}(x, y)$. What is the running time? The question is tricky because the way we defined it the simulation may *never* produce an answer. But suppose now that we allow only for r restarts and if all of them fail, $S^*(x)$ just outputs something arbitrary. Since b^* is random and independent of b, at every step the chance of requiring a restart is $1/2$; so the probability of requiring more than r restarts is 2^{-r}. To summarize, the distribution $S^*(x)$ is identical to $\text{view}_{A^*}(x, y)$ with probability at least $1 - 2^{-r}$ (over the internal randomness of S^*). It follows that in general, the distributions $S^*(x)$ and $\text{view}_{A^*}(x, y)$ are $(\infty, 2^{-r})$ computationally indistinguishable.

If we set $r = \log_2 1/\varepsilon$, we satisfy the requirement of (∞, ε) zero-knowledge. The simulator S^* runs $\log_2 1/\varepsilon$ rounds of the following: First, choose a random permutation π and a random b^*, compute I as $\pi(G)$ or $\pi(H)$, and simulate A^* using I and π as responses. The running time of this simulation is

$$(\log_2 1/\varepsilon) \cdot (\text{running time of } A^*(x)) + O(n \log n)$$

where $O(n \log n)$ is the time it takes to flip a random bit and generate a random permutation. We just proved the following:

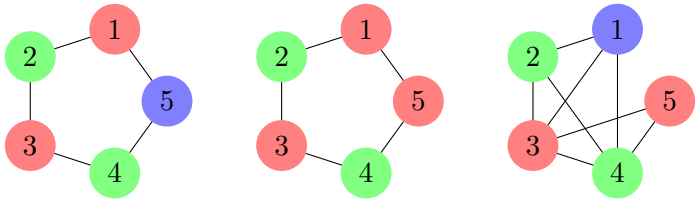
Theorem 2. *The interactive protocol (A, B) is an (∞, ε) zero-knowledge proof system for graph isomorphism with simulation overhead $oh(t) = O(t \cdot \log 1/\varepsilon) + O(n \log n)$, where n is the number of vertices of G and H.*

4 Hardness of 3-coloring

Was there anything special about graph isomorphism that allowed Bob to prove its existence in zero-knowledge? Looking at the protocol, it seems we used a lot of special properties about graphs, isomorphisms, and random permutations. Amazingly, it turns out that (assuming one-way functions exist) the answer is no: Any (correct) proof whose correctness can be checked reasonably fast can be verified in a zero-knowledge manner also reasonably fast!

To explain how this works we will need a bit of complexity theory. Complexity theory gives us many examples of proof relations that are "complete": All other proof relations are (in a well-defined sense we will see shortly) special cases of it. You may have already seen some such relations like SAT (boolean formula satisfiability) or $CLIQUE$. The one we will need today is called $3COL$ (for

graph 3-colorability). Given a graph G on n vertices, a *candidate 3-coloring* is an assignment of one of three colors $\{\mathtt{R}, \mathtt{G}, \mathtt{B}\}$ to the vertices of G. (If we order the vertices, a candidate 3-coloring can be represented as a string in $\{\mathtt{R}, \mathtt{G}, \mathtt{B}\}^n$ like RRGRGB.) We say the candidate 3-coloring *col* is *valid* for G if no two adjacent vertices are assigned the same color. Here are some examples of valid and invalid 3-colorings:



RGRGB
valid

RGRGR
invalid

BGRGR
invalid

(Notice the following difference between the last two examples: In the second example, this particular 3-coloring is invalid, but there exists another valid one. In the third example, no valid 3-coloring exists.)

The proof relation $3COL$ is defined as

$$(G, col) \in 3COL \text{ iff } col \text{ is a valid 3-coloring of } G.$$

Completeness of graph 3-coloring We now describe the sense in which $3COL$ is a "complete" proof relation. To do this we need the notion of a verifier for a proof relation.

A *verifier* for a proof relation R is an algorithm $V(x, y)$ that accepts if $(x, y) \in R$, and rejects otherwise. We already saw a verifier for graph isomorphism: It takes the graphs G and H and the purported isomorphism ϕ and checks that (u, v) is an edge in G if and only if $(\phi(u), \phi(v))$ is an edge in H for all pairs of vertices (u, v). A verifier for graph 3-coloring is no more difficult to design: The verifier takes as inputs G and *col*, goes over all edges of G, looks up the colors of the corresponding endpoints in *col*, rejects if a violation is ever detected, and rejects otherwise.

We will assume that the running time of V only depends on x and is independent of y.

One nice thing about the verifiers for both graph isomorphism and 3-coloring is that they are both *efficient*: The running time of the verifier is proportional to the size of the input (the statement plus the solution). We will only be interested in proof relations with efficient verifiers; after all, if we use the verifier as part of a two-party computation, it better be that Alice and Bob can run this verifier efficiently.

The graph isomorphism and 3-coloring questions look totally unrelated (apart from them both being questions about graphs, but this is a superficial connection). Amazingly, it turns out that they are very closely related to one another! Suppose we are given two isomorphic graphs and want to find the isomorphism. This may be hard, but imagine also that we have solved an unrelated problem: Given a 3-colorable graph G, we can find a 3-coloring for it. Then using our solution to the 3-coloring problem, we can also solve the graph isomorphism problem!

We will not give the details of this (see lecture notes 1 and 2 of CSCI5170) but let us describe the formal framework. In the case of graph isomorphism and 3-coloring, there exists a pair of algorithms (RSt, RPf) (statement and proof reduction) with the following properties:

1. RSt takes as inputs a pair of graphs (G, H) and produces as output a graph C. The graph C has a valid 3-coloring if and only if there is an isomorphism from G to H.

2. RPf takes as inputs a pair of graphs (G, H) and an isomorphism ϕ from G to H, and outputs a valid 3-coloring for the graph C.

The size[2] of C and the running times of RSt and RPf are $O((|G| + |H|)^2)$.

This example is an instance of a general theorem, which says that any proof relation (with an efficient verifier) can be "reduced" to $3COL$.

Theorem 3. *For every proof relation R with verifier V, there is a pair of algorithms (RSt, RPf), with running time quadratic in the running time of $V(x, \cdot)$, so that if $(x, y) \in R$ then $RPf(x, y)$ is a valid 3-coloring for the graph $RSt(x)$, and if $(x, y^*) \notin R$ for all y^*, then $RSt(x)$ does not have a valid 3-coloring.*

5 Zero-knowledge proofs for all theorems

Theorem 3 says that in some sense, $3COL$ is "the hardest" among all proof relations. So if we can design a zero-knowledge proof for $3COL$, then we can obtain one for all proof relations of interest, including the ones that come up in the protocol for honest-but-curious secure 2-party computation from last lecture.

We now describe this zero-knowledge proof for $3COL$. Alice's input is a graph G, Bob's input is a graph G together with a 3-coloring *col* for G, and Bob wants to convince alice that G is 3-colorable, without revealing any additional information (about *col* or anything else).

The key property of 3-colorings that we need (and the reason we chose to work with this problem in the first place) is that they have a very useful symmetry: Suppose we have a valid 3-coloring *col* of G and we permute the identities of the colors, say $\texttt{R} \to \texttt{B}$, $\texttt{G} \to \texttt{G}$, $\texttt{B} \to \texttt{R}$. Then the resulting coloring is still a valid 3-coloring for G.

There are six possible permutations on 3-colorings; let's denote them by $\pi_1, \ldots, \pi_6$. In addition to these permutations, we will need a commitment scheme (Com, Rev) (for trits instead of bits). Here is how the protocol works (when Alice and Bob are both honest):

Graph 3-Coloring Protocol: Alice and Bob are given a graph G on n vertices; in addition, Bob has a valid 3-coloring *col* for G.

B: Choose a random number $b \in \{1, \ldots, 6\}$ and applies the permutation π_b to every coordinate in his coloring. For every vertex v, compute a commitment $Com(K_v, col_b(v))$ to the color of vertex v in the coloring col_b. Send all the commitments to Alice (in a specific order).

A: Upon receiving the commitments $C_1, \ldots, C_n$, choose a random edge (u, v) of G and send this edge to Bob.

B: Upon receiving the edge (u', v'), send the keys $K_{u'}$ and $K_{v'}$ to Alice.

[2] This is the number of bits that it takes to describe C; for a graph, it roughly equals the number of vertices plus the number of edges.

A: Reveal the colors $R_u = Rev(K_{u'}, C_u)$ and $R_v = Rev(K_{v'}, C_v)$. If $R_u \neq R_v$, accept, otherwise reject.

The functionality of this protocol is clear: If *col* is a valid 3-coloring for G, then the commitments C_u and C_v will always refer to different colors when (u, v) is an edge of G, and upon revealing these commitments Alice accepts with probability 1.

Let us now argue soundness:

Claim 4. *Assume that (Com, Rev) is an (s, ε) secure commitment scheme. Let G be a graph with m edges that does not have a valid 3-coloring. Then for every B^* of size at most s',*

$$\Pr_{(A(G),B^*(G))}[A \text{ accepts}] \leq 1 - 1/m.$$

The probability $1 - 1/m + \varepsilon$ is not that low; it can be improved by repeating the protocol (in sequence) several times. One can also use alternative versions of graph coloring to achieve a lower success probability.

Proof. By the binding property of (Com, Rev), for every commitment C_v there is a unique key K_v and color $col'(v)$ such that $Rev(K_v, C_v) = col'(v)$. Since G is not 3-colorable, it must be that $col'(u) = col'(v)$ for at least one edge (u, v) of G, regardless of the algorithm B^*. If Alice happens to select this edge (u, v), then she detects an inconsistency with the coloring and rejects. Since there are m possible edges, this happens with probability at least $1/m$. □

Finally, we are left with zero-knowledge: We want to say that no matter what Alice tries to do, she can simulate her view from the interaction when *col* is a valid 3-coloring of G and Bob is honest. Let's get some intuition for this. At the end of the first round, Alice sees some commitments $C_1, \ldots, C_n$ to a color each. By the hiding property of commitments, Alice should have no idea what is the color in each commitment, so she can simulate this distribution by committing to *arbitrary* colors. Based on these commitments, Alice decides to send an edge (u, v) to Bob. (If Alice is malicious, this edge may not be random). Bob then sends the keys that would allow Alice to reveal the colors $col_b(u)$ and $col_b(v)$. From Alice's perspective, this is now just a pair of *random* different colors (since b is random): She could have created this distribution herself by committing to two random different colors in the first step!

This suggests the following simulation strategy $S^*(G)$. First, $S^*(G)$ guesses at random the edge (u', v') that Alice will query in round 2. It then chooses a random pair of colors (col^*_u, col^*_v) for u and v (there are six possibilities), and chooses a fixed color – say $col^*_w = \mathtt{R}$ – for all the other vertices w. To simulate the first round, it outputs the commitments $C_w = Com(K_w, col^*_b(w))$, where K_w is a random key.

For the second round, $S^*(G)$ uses $A^*(G)$ to find out its message (u, v) in the second round that is obtained upon seeing the commitments C_w. If $(u, v) \neq (u', v')$, the simulator fails and restarts (with fresh randomness). Otherwise it simulates the third round by the message (K_u, K_v), and it simulates A^* in the last round and outputs its answer.

Conditioned on $S^*(G)$ not failing, its output is distributed "almost" like $\text{view}_{A^*}(G, col)$. The only difference comes from the commitments: In the real interaction, Bob commits to the real coloring col_b, while the simulator creates a fake coloring col^*. The only way that these two colorings make it into the real and simulated views is via the commitments $Com(K_u, col_b(u))$ and $Com(K_u, col^*_b(u))$

for all vertices u. By assumption, any two such commitments are (s, ε) computationally indistinguishable. By a hybrid argument, it follows that conditioned on not failing, the real and simulated views are $(s - O(nt_{Com}), n\varepsilon)$ computationally indistinguishable, where t_{Com} is the time it takes to compute a commitment of one trit.

In each round, the probability of the simulation failing is $1/m$; so after repeating for r rounds, the simulation succeeds with probability $(1 - 1/m)^r$. Setting $r = 4m \log_2 1/\varepsilon$, we obtain a simulator $S^*(G)$ with simulation overhead $oh(t) = O(tm \log 1/\varepsilon + nt_{Com})$ which produces a simulated view that is $(s - O(nt_{Com}), (n+1)\varepsilon)$ indistinguishable from the real one.

Summarizing our analysis, we obtain the following theorem:

Theorem 5 (Zero-knowledge for 3-coloring)**.** *Assume (Com, Rec) is an (s, ε) secure commitment scheme for values $\{0, 1, 2\}$. Then the graph 3-coloring protocol is an $(s - O(nt_{Com}), (n+1)\varepsilon)$ zero-knowledge proof system with simulation overhead $oh(t) = O(tm \log 1/\varepsilon + nt_{Com})$ for $3COL$. Here t_{Com} is the circuit size of Com and n and m is the number of vertices and edges of G, respectively.*