CS4501 Cryptographic Protocols
Lecture 25: ZK for all of NP, Proofs of Knowledge, SLE

https://jackdoerner.net/teaching/#2026/Spring/CS4501

Big Claim: Zero-Knowledge Proofs

If you can prove a statement in such a way that a *computer* can verify the proof, then you can prove that statement without *conveying any knowledge*, beyond the fact that it is true.

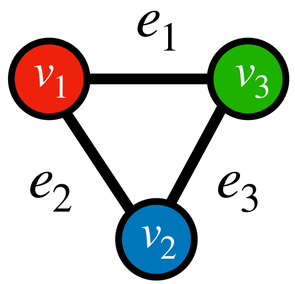
In particular, the verifier of the proof cannot even prove the statement to somebody else afterwards!

Put more simply: you can prove that something is true without revealing *why* it is true.

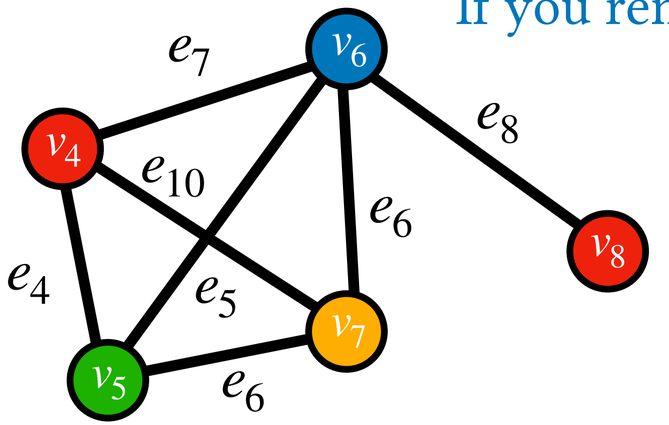
Coloring Graphs

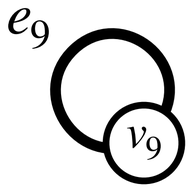
- We say that a graph is *k-colorable* if there exists a set of k colors such that:
  1. Every vertex is assigned one of the colors. This assignment is called a *coloring*.
  2. Every edge connects two vertices of *different* colors. If this is true the coloring is *proper*.
- Not every graph can be k-colored for every number k.
- Last class we demonstrated that I can *interactively prove* that a graph is k-colorable, without conveying any knowledge about the coloring.

Can you 2-color this one? **No.**
What about 3-coloring it? **Yes.**



How many colors for this one? **4**
If you remove e_{10}? **3**





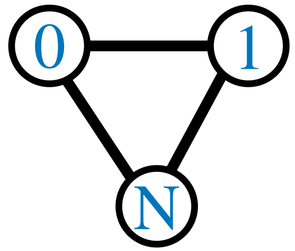
What about this one?
It can't even be 1-colored!

How to Prove *Anything* by Coloring a Graph

- Coloring graphs might seem like a silly problem, but it turns out to be an *extremely powerful one*. It's not just NP-hard, but NP-*complete*.
- If you give me *any* boolean circuit C that outputs exactly 1 bit, then I can construct a graph G such that G is 3-colorable *if and only if* there is at least one input x such that $C(x) = 1$.
- This includes the circuit C that takes as input an alleged proof of some particular statement and checks whether that proof is correct!
- Thus, if you can prove something to a circuit, *then you can prove it in zero-knowledge.*
- To simplify things, we will consider boolean circuits made up only of NOT ($\neg$) and OR ($\vee$) gates. This set of gates is complete for boolean functions (and in particular, AND is easy to construct using De Morgan's Law).

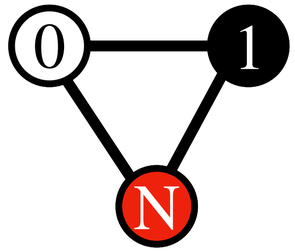
Step 1: The Colors of Truth and Falsehood

- We are going to use different vertices to *represent* different pieces of the circuit. Some vertices will represent constants such as the value 1 (true) and the value 0 (false). Some vertices will represent input variables. Some will represent the values of wires in the circuit.
- First, we will construct a piece of the graph that forces the 1 vertex the 0 vertex to be assigned different colors. When we build the rest of the graph, we will build it such that assigning any vertex the same color as the 1 vertex implies that the value of the variable associated with that vertex must be 1 (similarly for 0).
- To accomplish our task, we need to add a third value called *Neither* (N).



Step 1: The Colors of Truth and Falsehood

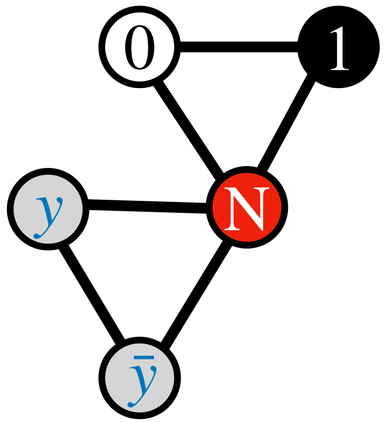
- We are going to use different vertices to *represent* different pieces of the circuit. Some vertices will represent constants such as the value 1 (true) and the value 0 (false). Some vertices will represent input variables. Some will represent the values of wires in the circuit.
- First, we will construct a piece of the graph that forces the 1 vertex the 0 vertex to be assigned different colors. When we build the rest of the graph, we will build it such that assigning any vertex the same color as the 1 vertex implies that the value of the variable associated with that vertex must be 1 (similarly for 0).
- To accomplish our task, we need to add a third value called *Neither* (N).



- I will use these 3 colors for examples.

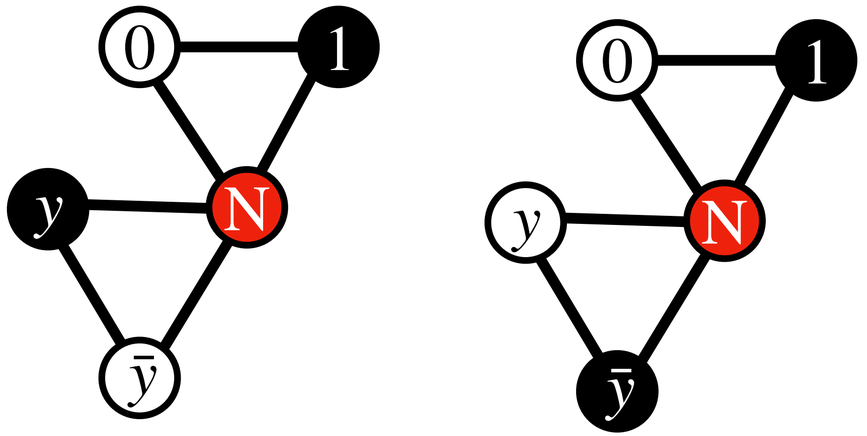
Step 2: Wires and NOT

- Suppose I want to represent a wire y in the circuit (which could be an input, output, or internal wire).
- I will create a vertex labeled y and a vertex labeled $\bar{y}$.
- By connecting both to N, I force them to have the colors 0 or 1. Thus they are both valid wires.
- By connecting them to each other, I force them to assume *different* colors. Thus $\bar{y} = \neg y$.



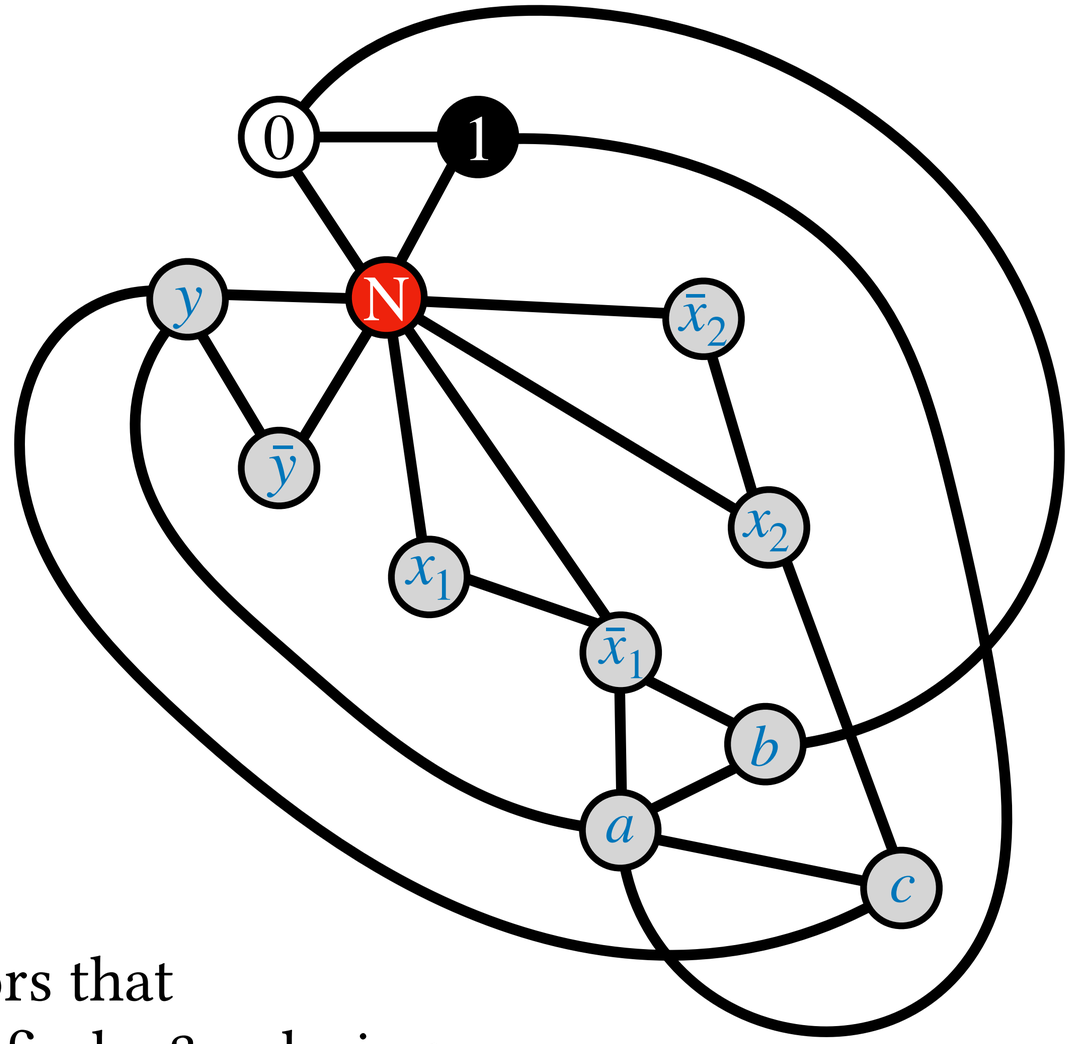
Step 2: Wires and NOT

- Suppose I want to represent a wire y in the circuit (which could be an input, output, or internal wire).
- I will create a vertex labeled y and a vertex labeled $\bar{y}$.
- By connecting both to N, I force them to have the colors 0 or 1. Thus they are both valid wires.
- By connecting them to each other, I force them to assume *different* colors. Thus $\bar{y} = \neg y$.
- For example, these are the possible colorings of the graph I have drawn so far:



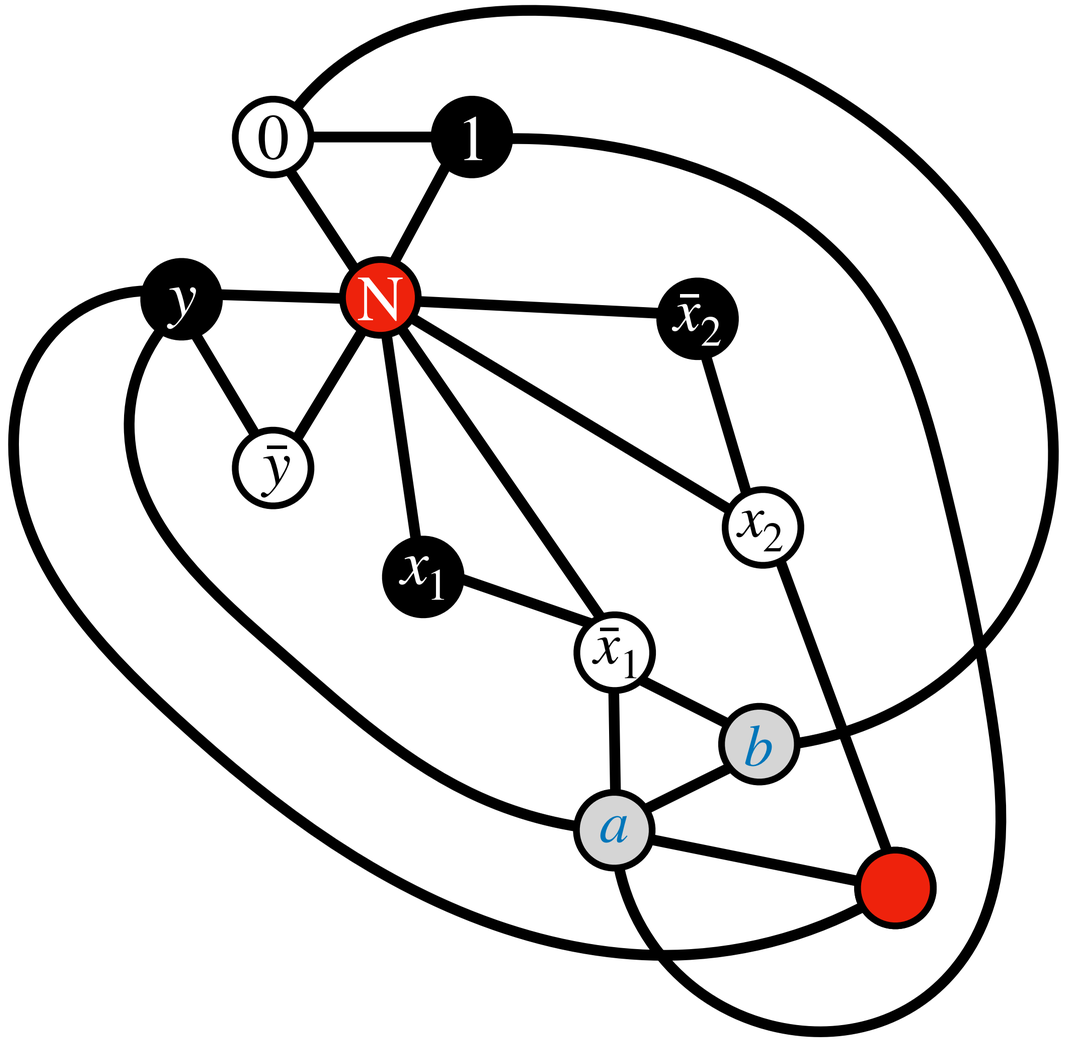
Step 3: OR gates

- Suppose I have represented the wires x_1, x_2, y using vertices as before, and I want to represent the OR gate $y = \bar{x}_1 \vee x_2$.
- I will create three extra vertices a, b, c that do not represent any wire.
- a and b will form a triangle with $\bar{x}_1$, and c will connect to x_2.
- Then we form a triangle with c, a, y.
- Finally, a connects to 1, and b connects to 0.
- Let's think about this. If we set $\bar{x}_1, x_2, y$ to colors that are valid for an OR gate, we should be able to find a 3-coloring for the rest of the graph. Otherwise, no proper 3-coloring should exist.



Step 3: OR gates

- We'll check $\bar{x}_1 = 0$, $x_2 = 0$ first.
- If $y = 1$, then no proper 3-coloring should exist.
- c must be red, because y is black and x_2 is white.
- Now a cannot be any color because it connects to all three!

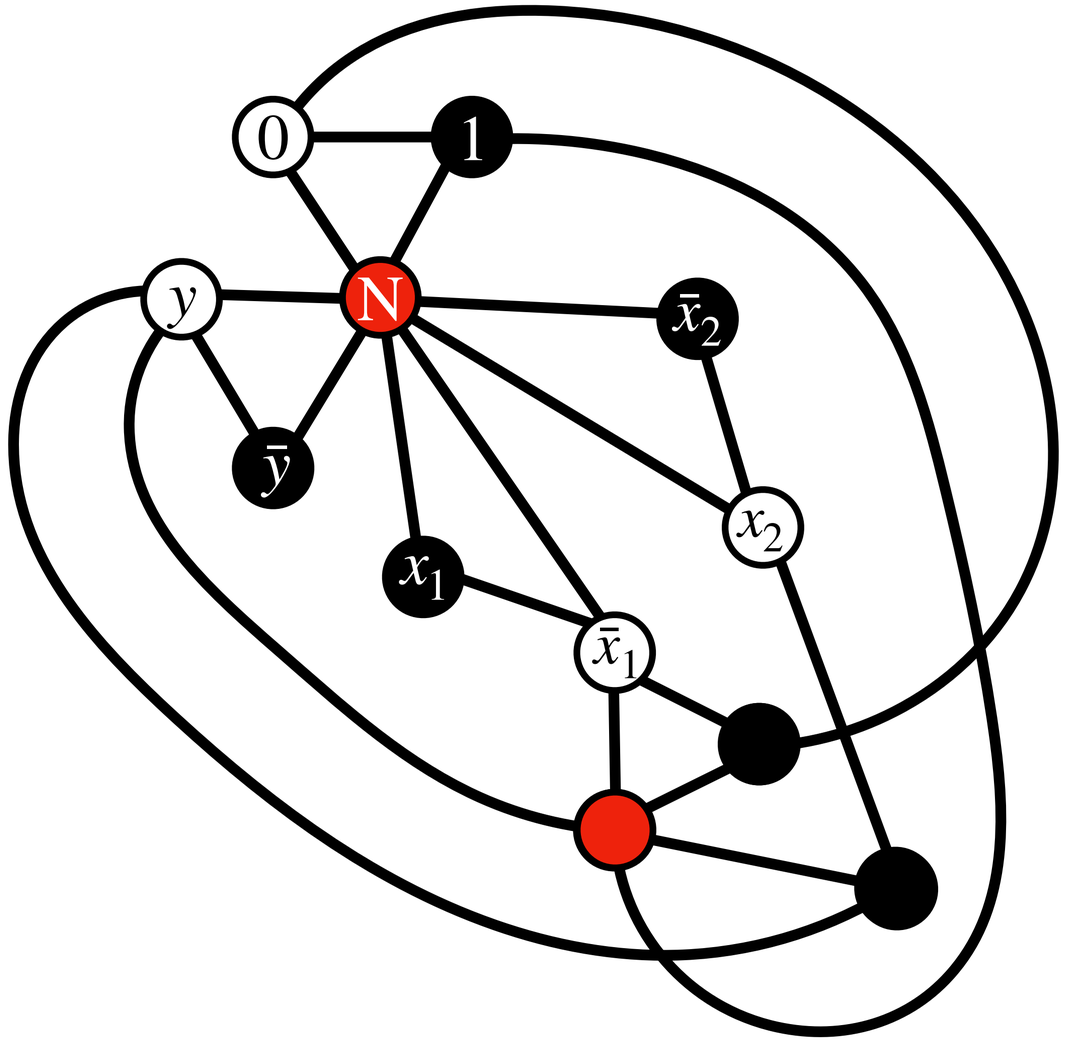


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- We'll check $\bar{x}_1 = 0$, $x_2 = 0$ first.
- If $y = 0$, then we should be able to find a 3 coloring.
- Here it is!

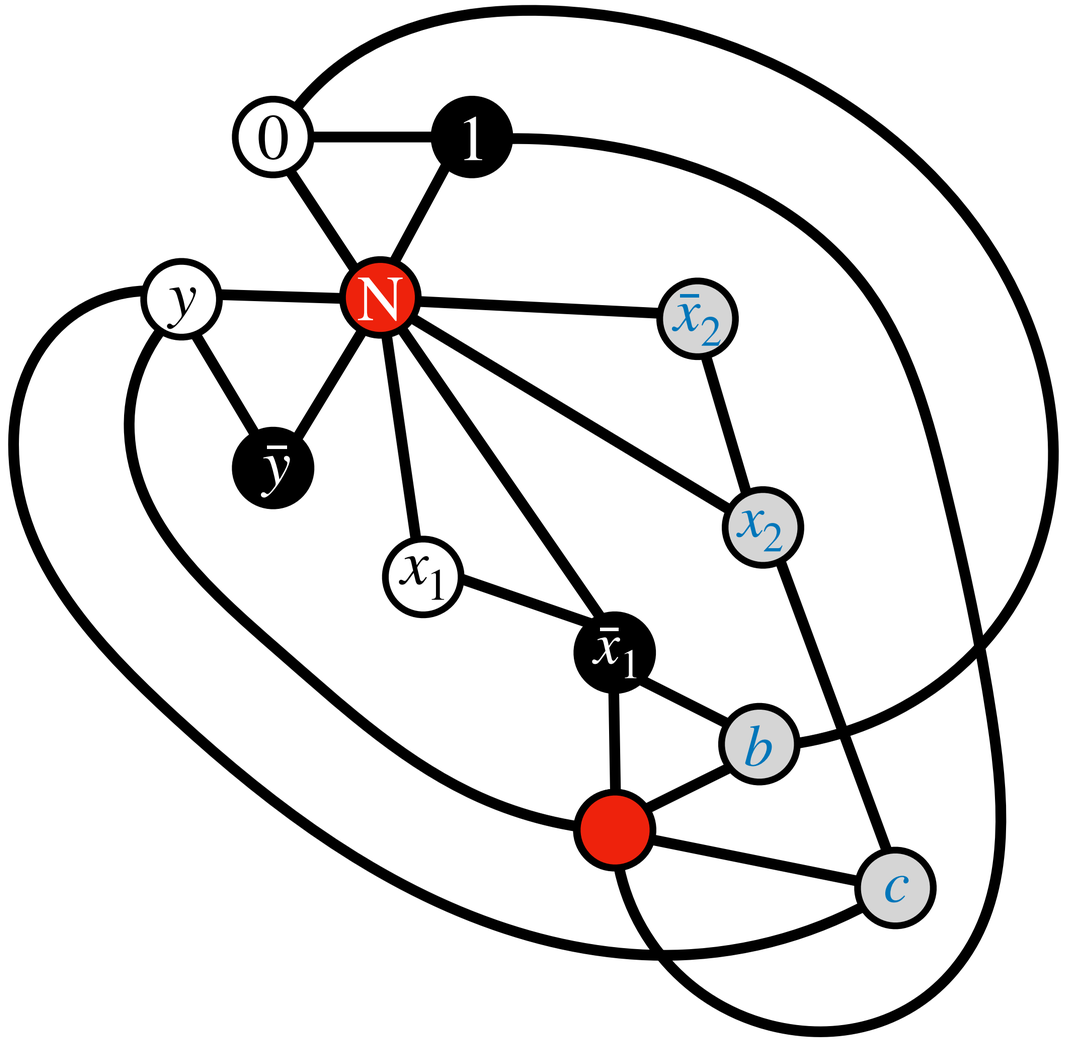


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- Similarly, suppose $\bar{x}_1 = 1,\ y = 0$.
- This should produce a contradiction regardless of the value of x_2.
- And indeed, a must be red, which implies that b is connected to all colors, and thus cannot be any color itself.

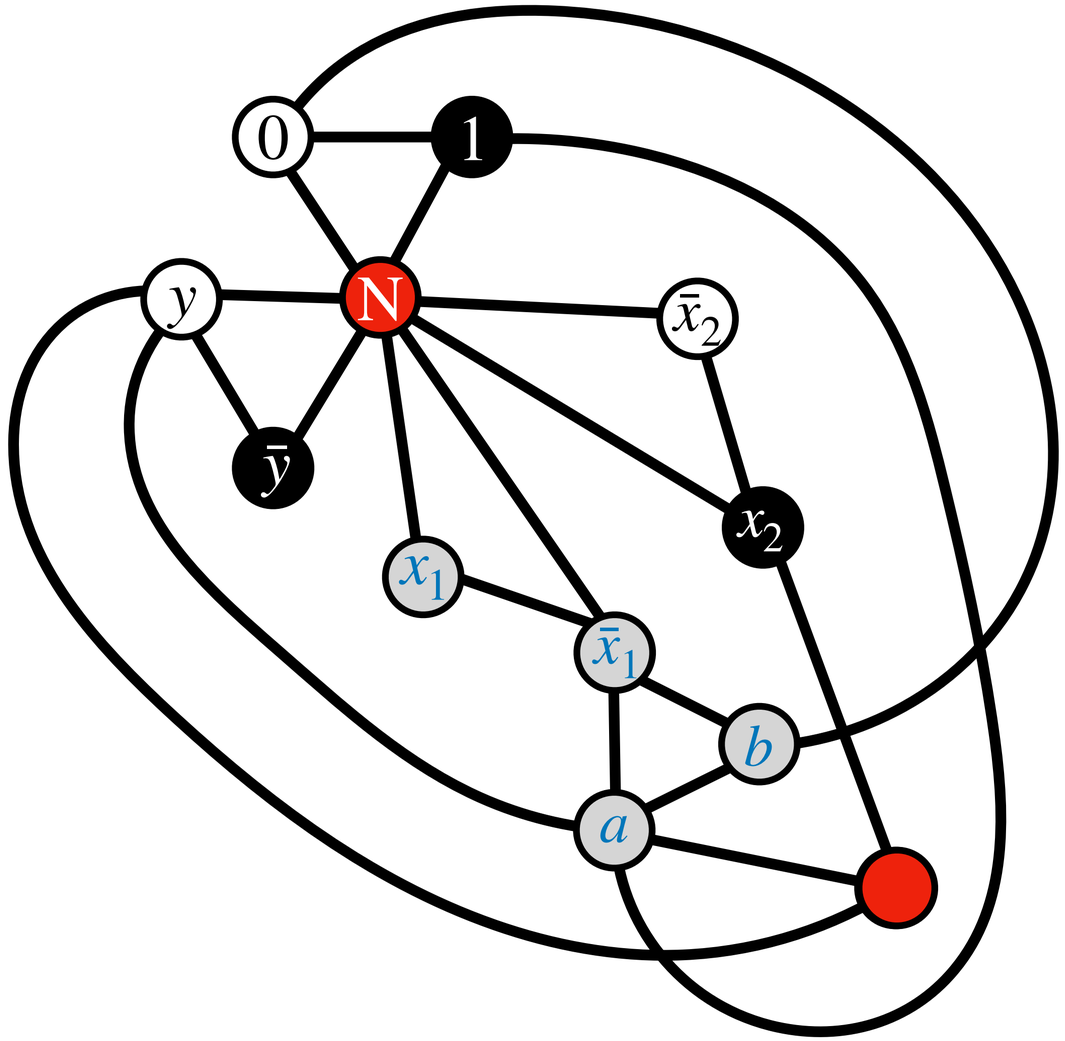


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- Next, suppose $x_2 = 1,\ y = 0.$
- This should produce a contradiction regardless of the value of $\bar{x}_1$.
- Now we see that c must be red, which implies that a is connected to all colors, and thus cannot be any color itself.

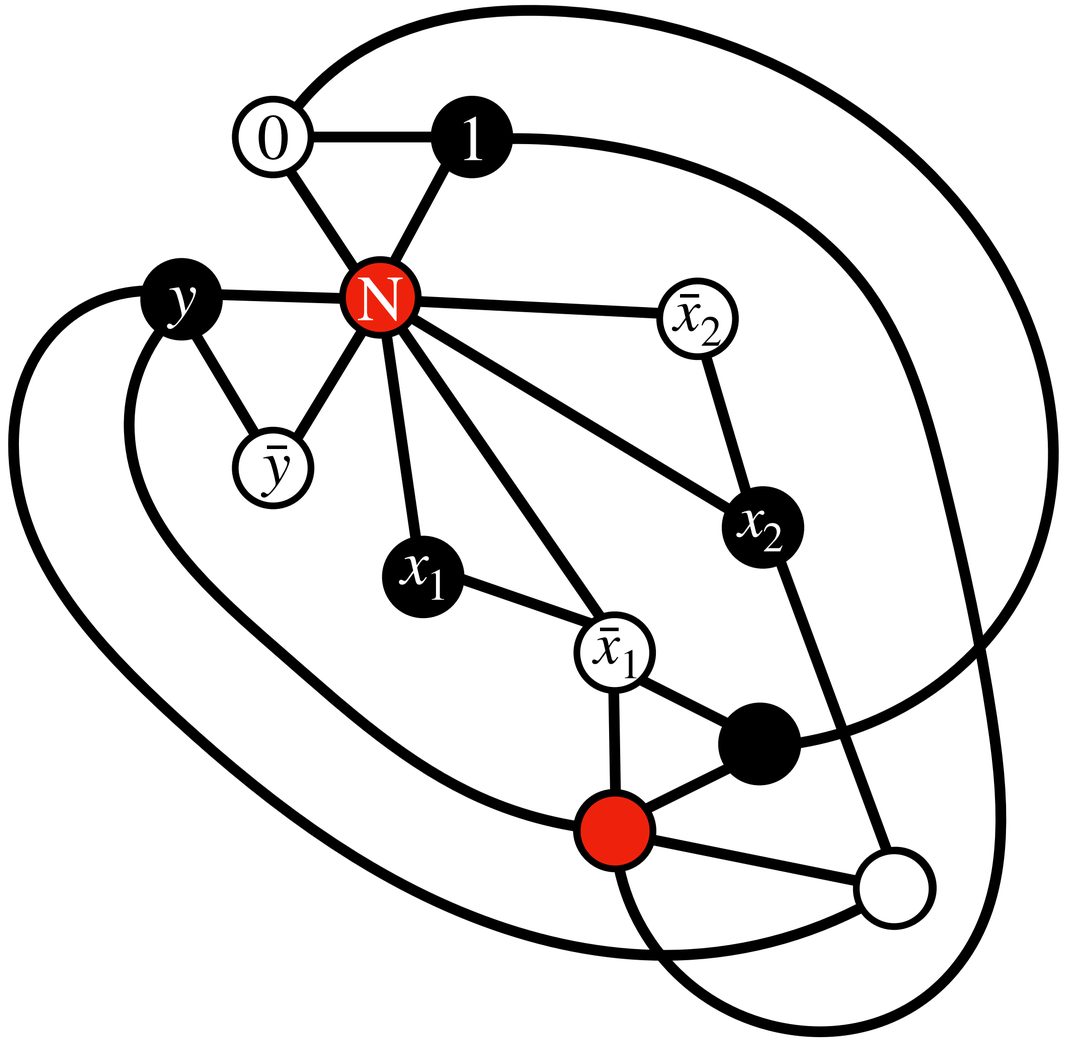


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- Finally, we must make sure that there exist proper 3-colorings for:
- $\bar{x}_1 = 0, x_2 = 1, y = 1$: **Yes**

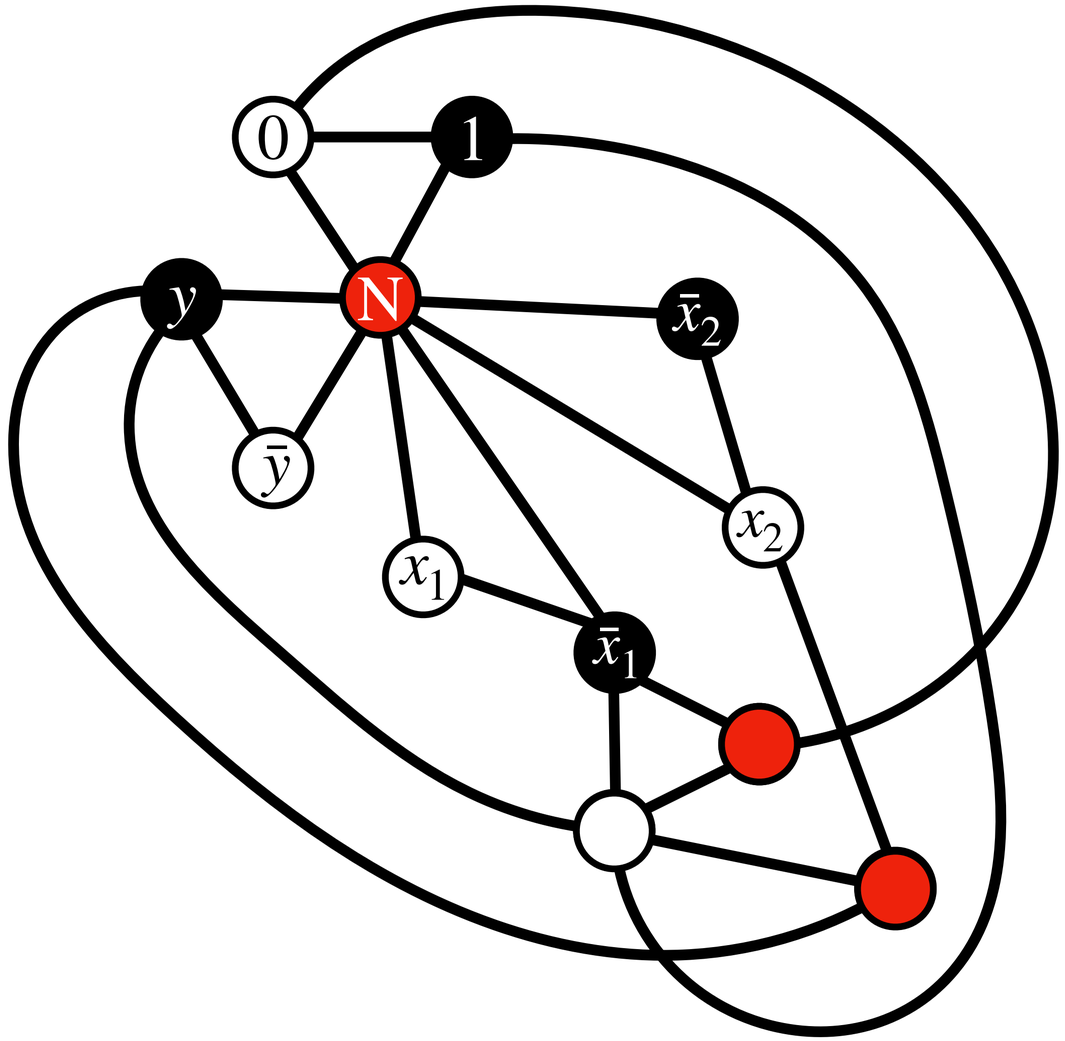


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- Finally, we must make sure that there exist proper 3-colorings for:
- $\bar{x}_1 = 0,\ x_2 = 1,\ y = 1$: **Yes**
- $\bar{x}_1 = 1,\ x_2 = 0,\ y = 1$: **Yes**

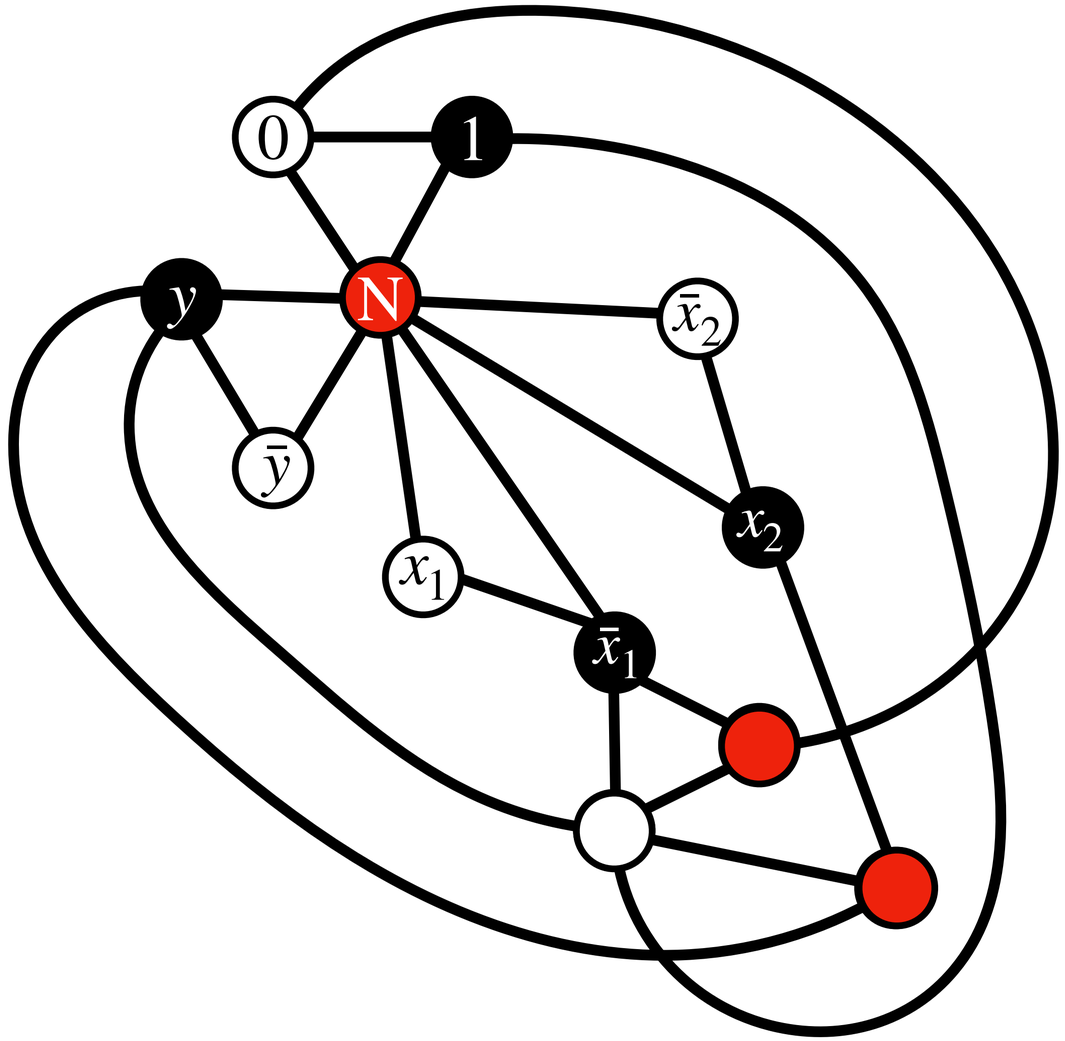


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 3: OR gates

- Finally, we must make sure that there exist proper 3-colorings for:
- $\bar{x}_1 = 0, x_2 = 1, y = 1$: **Yes**
- $\bar{x}_1 = 1, x_2 = 0, y = 1$: **Yes**
- $\bar{x}_1 = 1, x_2 = 1, y = 1$: **Yes**

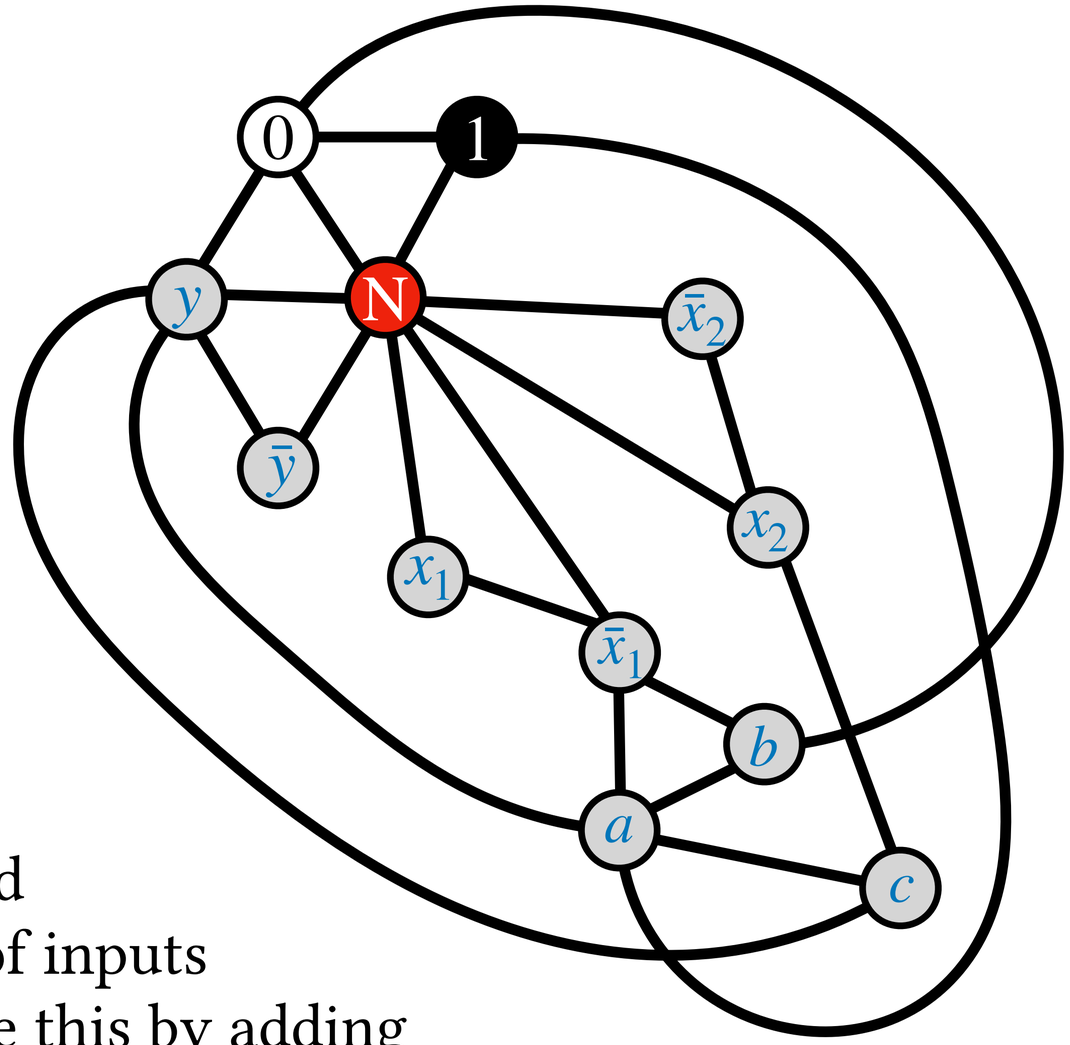


OR ($\vee$)

$\bar{x}_1$	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Step 4: True Statements

- So now we have a graph where the proper 3-colorings correspond *exactly* to all the rows in the truth table of the circuit $y = (\neg x_1) \vee x_2$.
- By extending this same methodology, you can add any number of NOT and OR gates in any topology. The size of the graph is linear in the circuit size.
- We're not quite done. We want this graph to be 3-colorable only if the statement checked by the circuit is *true*, i.e. only if there is a set of inputs such that the circuit outputs 1. We can enforce this by adding one final edge from the output vertex y to the 0 vertex. We're done!



Proving the Solution to Any NP Problem

- The problem of determining whether there exists an input such that some arbitrary circuit *is satisfied* (i.e. outputs 1) is called CSAT.
- Consider circuits of the form $(x_1 \vee x_3 \vee \neg x_7 \vee \ldots)$; that is, NOT gates can only be applied to the inputs. We will call such circuits *clauses*. The *length* of a clause is the number of variables.
- Now consider the form clause $\wedge$ clause $\wedge \ldots$ where the clauses are not the same, but may contain (some of) the same variables. Such circuits are *SAT instances*.
- Any SAT instance is also a CSAT instance. In 1966, Grigori Tseitin proved that for every circuit, there is an *equisatisfiable* SAT instance (i.e. one that is satisfiable if and only if the original circuit is) that is only a constant factor larger.

- In 1971 and 1973, Stephen Cook and Leonid Levin proved (independently) that SAT is an NP-complete problem. Thus for any *decision problem* in NP, there exists a polynomial-time algorithm that finds a graph that is 3-colorable if and only if the answer to the decision problem is "yes."

Formalizing Languages

Definition 1: A *Decidable* Language $L \subseteq \{0,1\}^*$ is a set of strings such that there exists a deterministic decider (i.e. membership verifier) algorithm R_L with the following properties:

- R_L always eventually halts, although it may take an unbounded amount of time to do so.
- $\forall x \in L \quad \exists w \in \{0,1\}^*$ such that $R_L(x, w) = 1$. We call x a *statement* in the language and w a *witness* for the membership of x.
- $\forall x \notin L \quad \forall w \in \{0,1\}^*$ we have $R_L(x, w) = 0$. In other words, statements are in the language if and only if they have a witness.

Examples: a 3-coloring witnesses the language of 3-colorable graphs. $a, b \in \mathbb{Z}_q$ witness membership in the language of Diffie-Hellman Tuples $\{(g, g^a, g^b, g^{a \cdot b})\}_{a,b \in \mathbb{Z}_q}$.

Distressingly, there are also *undecidable* languages. You'll learn about them in a theory of comp class!

Definition 2: An *NP Language* $L \subseteq \{0,1\}^*$ is a decidable language for which R_L is a polynomial-time algorithm (or equivalently, a family of circuits with polynomial *size* relative to their input lengths), and $\forall x \in L \quad \exists w \in \{0,1\}^*$ such that $|w| \in O(\text{poly}(|x|))$ and $R_L(x, w) = 1$.

In other words, membership in any NP language can be *verified* by a poly-time algorithm.

Formalizing Zero-Knowledge Proofs

Definition 3: An *Interactive Proof System* (IPS) for language $L \subseteq \{0,1\}^*$ is a two-party protocol involving a (potentially unbounded, randomized) prover P and a PPT verifier V. It has two properties:

- **Completeness:** For every $x, w \in \{0,1\}^*$ such that $R_L(x,w) = 1$, if $P(x,w)$ interacts honestly with $V(x)$, then $\Pr[\mathsf{OUTPUT}_{V(x)} = 1] = 1$.
- **Soundness:** There exists a *noticeable* function δ such that for $x \notin L$ and every (possibly unbounded) malicious P^*, when P^* interacts with $V(x)$ we have $\Pr[\mathsf{OUTPUT}_{V(x)} = 1] \leq 1 - \delta(|x|)$.

We call $1 - \delta(|x|)$ in the above definition the *soundness error*.

If soundness holds only against provers that are PPT with respect to $|x|$, then the protocol is an interactive *argument* system.

Definition 4: An IPS is a *Zero-Knowledge Proof* (ZKP) if for every malicious PPT verifier V^*, there exists a PPT simulator algorithm Sim such that $\{\mathsf{VIEW}_{V^*}\}_{x \in L} \approx_c \{\mathsf{Sim}(x)\}_{x \in L}$.

Notice that this is a *one-sided* simulation definition: we don't care (yet) about simulating corrupt provers, nor about simulating corrupt verifiers when $x \notin L$.

Notice also, we haven't yet defined a functionality for ZKPs! If we did, it would always output 1 to the verifier when $x \in L$, which means that it would always output 1 in the cases we simulate above.

Formalizing Zero-Knowledge Proofs

Definition 5: An IPS is *Honest Verifier Zero-Knowledge Proof* (HVZK) if for every *semi-honest* PPT verifier V^*, there exists a PPT simulator algorithm Sim such that $\{\mathsf{VIEW}_{V^*}\}_{x \in L} \approx_c \{\mathsf{Sim}(x)\}_{x \in L}$.

"Honest" in the context of ZK is equivalent to "semi-honest" in the context of MPC. Yes, this is annoying. I will mostly use "semi-honest" in class to keep things clear.

Let $L_{\mathsf{CSAT}} = \{C \in \text{Circuits} : \exists w \in \{0,1\}^* \text{ s.t. } C(w) = 1\}$ be the language of all circuits that are *satisfiable* (i.e. that output 1 on at least one input). In any introductory theory of computation class, you will prove that L_{CSAT} is NP-complete, meaning that the task of deciding any NP language can be can be reduced to the task of deciding L_{CSAT}.

In last lecture and this lecture, we demonstrated:

Theorem 1: Assuming perfectly-binding commitment schemes exist, there is an HVZK IPS for L_{G3C}, with soundness error $1 - 1/|E|$, where $|E|$ is the number of edges in the instance.

Theorem 2: deciding L_{CSAT} can be reduced deciding L_{G3C}, the language of 3-colorable graphs.

By corollary, there must be an HVZK IPS for every NP language. In other words *anything you can prove to a computer, you can prove in zero knowledge.*

Formalizing Zero-Knowledge Proofs

In last lecture and this lecture, we demonstrated:

Theorem 1: Assuming perfectly-binding commitment schemes exist, there is an HVZK IPS for L_{G3C}, with soundness error $1 - 1/|E|$, where $|E|$ is the number of edges in the instance.

Theorem 2: deciding L_{CSAT} can be reduced deciding L_{G3C}, the language of 3-colorable graphs.

By corollary, there must be an HVZK IPS for every NP language. In other words *anything you can prove to a computer, you can prove in zero knowledge.*

In Grad Crypto, we will prove that if one-way functions exist, then there is a ZKP (with security against malicious verifiers and unbounded provers) for every NP-language. This proof requires an is *intense* pebbling game with many hybrid experiments.

In this class, I'll ask you an easier **question:** using only the tools we have already introduced, can you think of a way to turn our HVZK IPS for L_{G3C} a ZK *argument* with security against malicious verifiers?

Answer: if we use *equivocable* commitments in the CRS model, then the simulator can simulate all of the vertex commitments, and when V^* requests that a pair of vertices be opened, the simulator can equivocate that pair to open to two distinct, random colors.

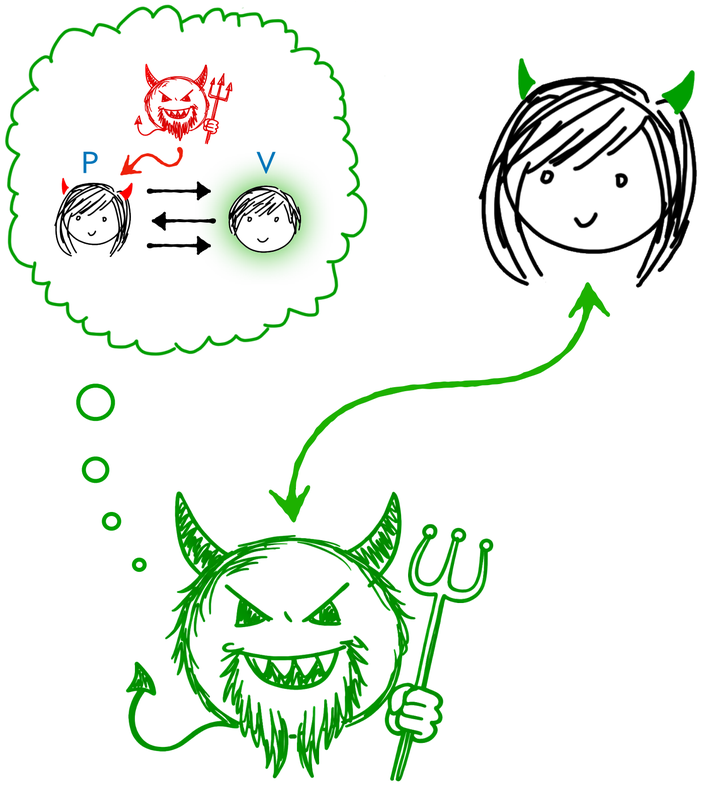
Toward Improving Soundness

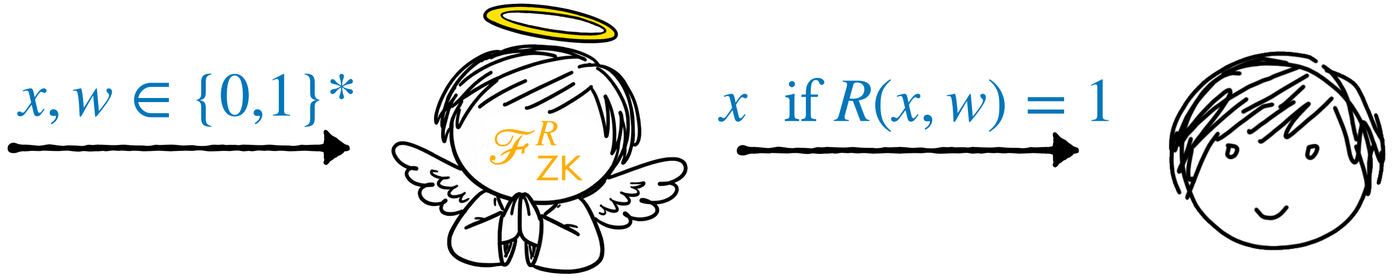
- Perhaps you noticed: our soundness error is *huge*. A malicious prover is very likely to convince us of false theorems. As we saw, we can improve this by *repetition*.
- In order to repeat a proof, we need a *composition theorem* that preserves soundness and ZK.
- In grad crypto, we prove that there exists a *sequential* composition theorem with these guarantees.
- We can also show that in general, zero-knowledge against malicious verifiers is *not* preserved under *parallel* composition. On the other hand, HVZK *is* preserved.
- **Q:** *why can't we use our existing composition theorem?*
- **A:** *it only applies to protocols that realize functionalities, of course!*
- Where we stand: Our definition of zero-knowledge only demands *one-sided* simulation, so our ZKPs won't realize any functionality in the presence of a corrupt prover.
- To fix this, we will design a *protocol compiler* that takes HVZK ZKPs of a certain special form, and modifies them to amplify soundness and add simulation against malicious provers and verifiers.

The Zero-Knowledge Functionality

- At a high level, we want a functionality that allows the prover to input some statement, and the verifier receives and indication that the statement is a member of some agreed language.
- Of course, deciding a statement with no other information might take a super polynomial amount of time (and an arbitrarily language could even be undecidable). To avoid this, we will give the functionality access to a membership-verifier algorithm, and we will insist that the prover supply the witness for any statement it wishes to prove.
- For any NP-Language L, let R be the associated polynomial-time membership-verifier algorithm (which exists by definition).
- The prover inputs a statement x and witness w. The verifier learns the statement, and it learns whether or not w witnesses $x \in L$.

Toward Proofs of Knowledge





- If the prover is corrupt, then $\mathcal{S}$ must somehow extract a witness.
- This (potentially) contradicts the zero-knowledge property of the ZKP, which suggests that we must give the simulator a trapdoor.
- Moreover, for some languages, deciding membership is *easy*, but finding a witness for membership might be hard.
- For example, consider prime modulus p and $g \in \mathbb{Z}_p^*$. Under modular multiplication, $\langle g \rangle \leq \mathbb{Z}_p^*$. Let $|\langle g \rangle| = q$. Lagrange's Thm implies $h \in \langle g \rangle \iff h^q \bmod p = 1$, but the witness for $h \in \langle g \rangle$ is the discrete logarithm of h, which is (assumedly) hard to compute!
- Our proof must guarantee that the prover *actually knows a witness to extract.* Interactive proofs do not generally guarantee this!

Proofs of Knowledge

We can formalize this notion by introducing a *stronger* soundness condition, which guarantees extraction (with sufficient probability) from any prover that can convince a verifier.

Definition 6: An IPS for language L is a *Proof of Knowledge* (PoK) if:

1. Non-triviality: There is some prover algorithm P that convinces V that $x \in L$ with probability 1, given x, w such that $R_L(x, w) = 1$ as input.
2. Knowledge-Soundness: There exists an additional PPT algorithm Ext and a negligible function ε such that for ever malicious prover algorithm P^* and every $x \in \{0,1\}^*$, we have

$$\Pr\left[R_L(x, \hat{w}) = 1 : \hat{w} \leftarrow \mathsf{Ext}^{P^*}(x)\right] \geq \Pr[\mathsf{OUTPUT}_{V(x)} = 1] - \varepsilon(|x|).$$

   In other words, the probability that Ext can extract a valid witness for x from P^* is *almost* as high as the probability that P^* can convince V that $x \in L$.

There are many ways to achieve extraction. Often, they involve *rewinding the prover* (which is just an algorithms that the extractor runs as a subroutine; therefore it can be checkpointed and resumed).

In our model, $\mathcal{S}$ needs to do the extraction, but it *can't* rewind a corrupt prover, because $\mathcal{A}$ might be interacting with the distinguisher $\mathcal{D}$, and $\mathcal{D}$ is *not* a subroutine of the simulator.

Sigma Protocols

We will focus on a special kind of ZKP with a special soundness property, and show that our protocol compiler can extract from it *without rewinding*. We call this *straight-line extraction.*

Definition 7: A Sigma Protocol is an IPS with the following 3-message structure:

1. The prover sends a statement x and a commitment message a to the verifier.
2. The verifier samples a challenge e *uniformly* from some set E, and sends it to he prover.
3. The prover sends a response z to the challenge.

Definition 8: A sigma protocol is said to be n-special-sound if there exists an additional PPT algorithm $\mathsf{Reconstruct}$ such that takes as input $x, a, \{e_i, z_i\}_{i\in[n]}$, and achieves the following property:

If V outputs 1 given the transcript (x, a, e_i, z_i) for every $i \in [n]$, and $e_i \neq e_j$ for every $i, j \in [n]$, then $\mathsf{Reconstruct}(x, a, \{e_i, z_i\}_{i\in[n]})$ outputs $\hat{w}$ such that $R(x, \hat{w}) = 1$. Otherwise, $\mathsf{Reconstruct}$ outputs $\perp$.

That is, $\mathsf{Reconstruct}$ outputs a witness given n different accepting proofs with the same first message.

Definition 9: A sigma protocol is *Special Honest Verifier Zero-Knowledge* (SHVZK) if for every semi-honest PPT V^*, there exists a PPT Sim that simulates given a *specific* challenge e in advance. That is, $\{\mathsf{VIEW}_{V^*}\}_{x\in L} \approx_c \{\mathsf{Sim}(x, e) : e \leftarrow E_x\}_{x\in L}$, where E_x is the challenge set associated with x.

You Have Already Seen Sigma Protocols!

Think back to the ZKP for L_{G3C} that we demonstrated last class. *Did it have these properties?*

Claim 1: It was a sigma protocol:

1. Prover sends the graph, permutes the coloring, and commits to the coloring vertex-by-vertex.
2. Verifier chooses a uniform edge (the challenge) from the set of all edges.
3. Prover opens the commitments for the vertices at the ends of that edge (response).

Claim 2: It was $|E|$-special-sound (where E is the set of edges, which is also the challenge set):

If you have *one* commitment to a coloring, and you open the vertices at the end of *every* edge, then you know the colors of all of the vertices!

Furthermore, if we derived our L_{G3C} instance from an L_{CSAT} instance C, then $|E| \in O(|C|)$.

Claim 3: It was SHVZK:

Remember the simulation strategy: first, choose an edge randomly, then commit to random, distinct colors at either end of the chosen edge (and commit to some fixed color everywhere else), and then, open the commitments to the edge that was chosen.

Our Protocol Compiler

Theorem 3: Let ℓ be a polynomial and let L be a language with membership-verifier algorithm R. If there exists an $\ell(|x|)$-special-sound SHVZK sigma protocol for L, and there exists an *extractable* commitment scheme, then there exists a protocol in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model that realizes $\mathcal{F}_{\mathsf{ZK}}^{R}$.

Proof: Let

- $(a, s) \leftarrow \mathsf{Prove}_1(x, w)$ and $z \leftarrow \mathsf{Prove}_3(s, x, w, a, e)$ be randomized algorithms that sample the prover's messages in the sigma protocol, and $\mathsf{Verify}(x, a, e, z)$ be verifier's output function.
- E_x be the challenge set for statement x, of size exactly $\ell(|x|)$. If the sigma protocol has a larger challenge set, then an arbitrary subset of size $\ell(|x|)$ can be chosen.
- Sim be the SHVZK simulator.
- $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathsf{TGen}, \mathsf{Ext})$ be the extractable commitment scheme.
- $\rho = \ell(|x|) \cdot \kappa$ be a *repetition count*. This is the number of times we need to repeat the proof to achieve negligible soundness error.

Now we are ready to see the protocol. At a high level, we will run ρ proofs in parallel. The verifier will commit to its challenge before the prover sends its first message, and the prover will commit to responses for *all* challenges before seeing which challenge it must answer.

Our Protocol Compiler

receives inputs $1^\kappa, x, w$

if $R(x, w) = 0$, halt immediately without sending messages. Otherwise…

1^κ

k

1^κ

k

receives input 1^κ

x

$\forall i \in [\rho],\ e_i \leftarrow E_x$

$\forall i \in [\rho],\ (c_i^e, d_i^e) \leftarrow \text{Com}_k(e_i)$

$c_1^e, \ldots, c_\rho^e$

$\forall i \in [\rho],\ a_i \leftarrow \text{Prove}_1(x, w)$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ z_{i,\hat{e}} \leftarrow \text{Prove}_3(x, w, a_i, \hat{e})$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ (c_{i,\hat{e}}^z, d_{i,\hat{e}}^z) \leftarrow \text{Com}_k(z_{i,\hat{e}})$

$a_1, \ldots, a_\rho, \left\{ c_{1,\hat{e}}^z, \ldots, c_{\rho,\hat{e}}^z \right\}_{\hat{e} \in E_x}$

$d_1^e, \ldots, d_\rho^e$

$\forall i \in [\rho],\ e_i := \text{Open}_k(c_i^e, d_i^e)$

$d_{1,e_1}^z, \ldots, d_{\rho,e_\rho}^z$

$\forall i \in [\rho],\ z_{i,e_i} := \text{Open}_k(c_{i,e_i}^z, d_{i,e_i}^z)$

outputs x if :

$\forall i \in [\rho],\ \text{Verify}(x, a_i, e_i, z_{i,e_i}) = 1$

Intuition: Simulating Corrupt Verifiers

receives input 1^κ

1^κ →

← 1^κ

k →

if $\mathcal{S}$ can extract e_i from c_i^e in advance,
then it can use the SHVZK simulator Sim to
simulate a_i, z_{i,e_i} without knowledge of w.

x →

$\forall i \in [\rho],\ e_i \leftarrow E_x$

$\forall i \in [\rho],\ (c_i^e, d_i^e) \leftarrow \mathsf{Com}_k(e_i)$

← $c_1^e, \ldots, c_\rho^e$

$\forall i \in [\rho],\ a_i \leftarrow \mathsf{Prove}_1(x, w)$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ z_{i,\hat{e}} \leftarrow \mathsf{Prove}_3(x, w, a_i, \hat{e})$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ (c_{i,\hat{e}}^z, d_{i,\hat{e}}^z) \leftarrow \mathsf{Com}_k(z_{i,\hat{e}})$

$a_1, \ldots, a_\rho, \left\{ c_{1,\hat{e}}^z, \ldots, c_{\rho,\hat{e}}^z \right\}_{\hat{e} \in E_x}$ →

← $d_1^e, \ldots, d_\rho^e$

$\forall i \in [\rho],\ e_i := \mathsf{Open}_k(c_i^e, d_i^e)$

$d_{1,e_1}^z, \ldots, d_{\rho,e_\rho}^z$ →

$\forall i \in [\rho],\ z_{i,e_i} := \mathsf{Open}_k(c_{i,e_i}^z, d_{i,e_i}^z)$

outputs x if :

$\forall i \in [\rho],\ \mathsf{Verify}(x, a_i, e_i, z_{i,e_i}) = 1$

Simulating Corrupt Verifiers

$(k, t) \leftarrow \mathsf{TGen}(1^\kappa)$

1^κ

k

upon receiving x from $\mathcal{F}_{\text{ZK}}^R$

x

$c_1^e, \ldots, c_\rho^e$

$\forall i \in [\rho],\ e_i := \mathsf{Ext}_t(c_i^e)$

$\forall i \in [\rho],\ a_i, z_{i,e_i} \leftarrow \mathsf{Sim}(x, e_i)$

$\forall i \in [\rho],\ \forall \hat{e} \neq e_i,\ z_{i,\hat{e}} := z_{i,e_i}$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ (c_{i,\hat{e}}^z, d_{i,\hat{e}}^z) \leftarrow \mathsf{Com}_k(z_{i,\hat{e}})$

$a_1, \ldots, a_\rho, \left\{ c_{1,\hat{e}}^z, \ldots, c_{\rho,\hat{e}}^z \right\}_{\hat{e} \in E_x}$

$d_1^e, \ldots, d_\rho^e$

$\forall i \in [\rho],\ e_i := \mathsf{Open}_k(c_i^e, d_i^e)$

$d_{1,e_1}^z, \ldots, d_{\rho,e_\rho}^z$

Simulating

$(k, t) \leftarrow \mathsf{TGen}(1^\kappa)$

upon receiving x from $\mathcal{F}_{\mathsf{ZK}}^R$

$\forall i \in [\rho],\ e_i := \mathsf{Ext}_t(c_i^e)$

$\forall i \in [\rho],\ a_i, z_{i,e_i} \leftarrow \mathsf{Sim}(x, e_i)$

$\forall i \in [\rho],\ \forall \hat{e} \neq e_i,\ z_{i,\hat{e}} := z_{i,e_i}$

$a_1,$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ (c_{i,\hat{e}}^z, d_{i,\hat{e}}^z) \leftarrow \mathsf{Com}_k(z_{i,\hat{e}})$

$\forall i \in [\rho],\ e_i := \mathsf{Open}_k(c_i^e, d_i^e)$

In order to prove that this simulation is indistinguishable from the real world, we use a sequence of hybrid experiments.

In the first hybrid, we generate k within $\mathcal{F}_{\mathsf{CRS}}$ using $(k, t) \leftarrow \mathsf{TGen}(1^\kappa)$ instead of $k \leftarrow \mathsf{Gen}(1^\kappa)$. This is indistinguishable from the real world by reduction to the extractable security of the commitment scheme.

Next, we replace each $z_{i,\hat{e}} \leftarrow \mathsf{Prove}_3(x, w, a_i, \hat{e})$ where $\hat{e} \neq e_i$ with z_{i,e_i}, one hybrid at a time. Each is indistinguishable from its predecessor by reduction to the hiding of the commitment.

Finally, we replace each $a_i \leftarrow \mathsf{Prove}_1(x, w)$ and $z_{i,e_i} \leftarrow \mathsf{Prove}_3(x, w, a_i, e_i)$ with $a_i, z_{i,e_i} \leftarrow \mathsf{Sim}(x, e_i)$, one hybrid at a time. Each is indistinguishable from its predecessor by reduction to the SHVZK property of the sigma protocol.

Now we are in the ideal world.
We passed through $\ell(|x|) \cdot \rho + 1$ hybrids.

Intuition: Simulating Corrupt Provers

receives inputs $1^\kappa, x, w$

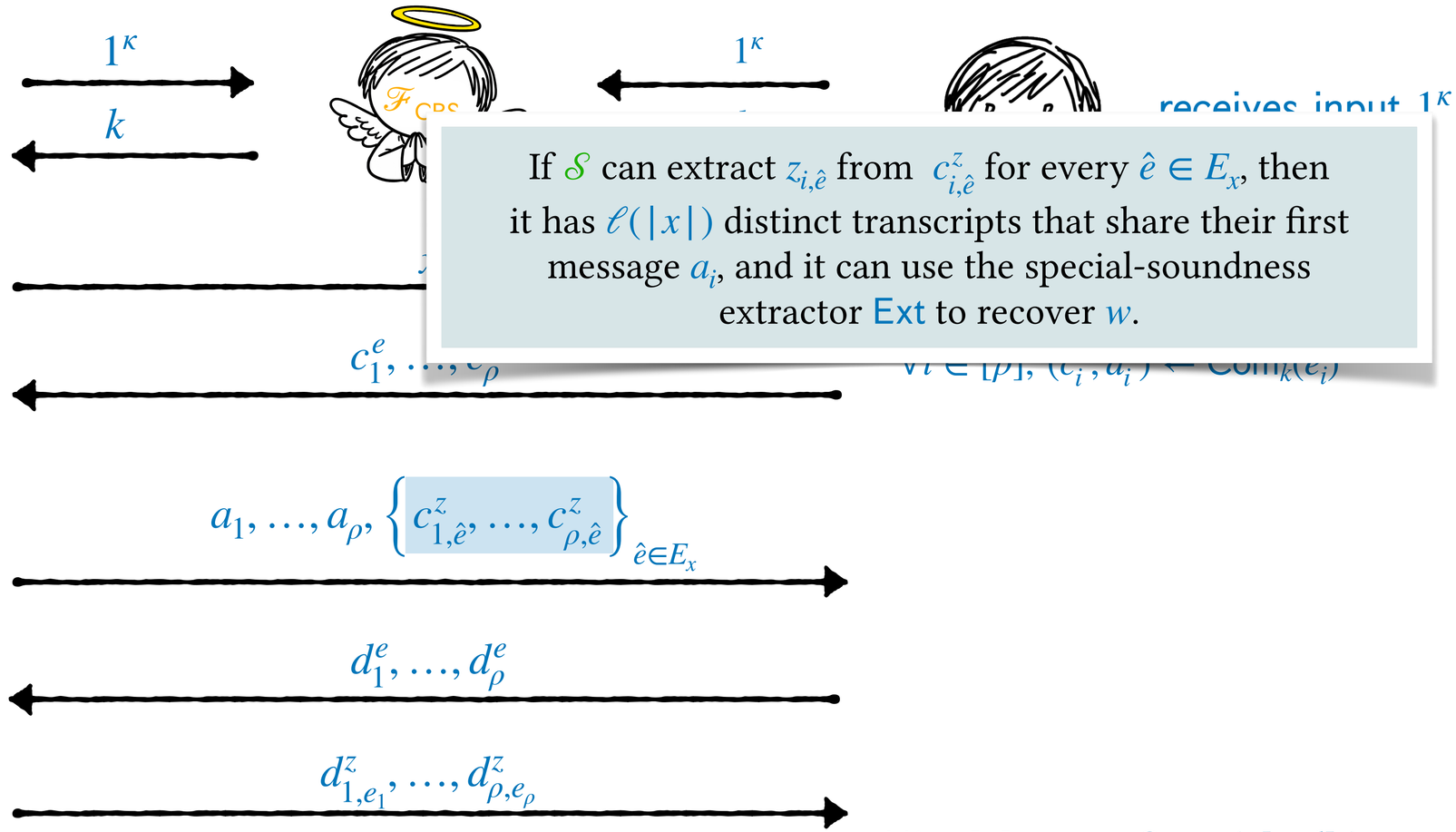
if $R(x, w) = 0$, halt immediately without sending messages. Otherwise…

$\forall i \in [\rho],\ a_i \leftarrow \mathsf{Prove}_1(x, w)$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ z_{i,\hat{e}} \leftarrow \mathsf{Prove}_3(x, w, a_i, \hat{e})$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ (c^z_{i,\hat{e}}, d^z_{i,\hat{e}}) \leftarrow \mathsf{Com}_k(z_{i,\hat{e}})$

$\forall i \in [\rho],\ e_i := \mathsf{Open}_k(c^e_i, d^e_i)$



If $\mathcal{S}$ can extract $z_{i,\hat{e}}$ from $c^z_{i,\hat{e}}$ for every $\hat{e} \in E_x$, then it has $\ell(|x|)$ distinct transcripts that share their first message a_i, and it can use the special-soundness extractor Ext to recover w.

receives input 1^κ

$\forall i \in [\rho],\ (c_i, a_i) \leftarrow \mathsf{Com}_k(e_i)$

$\forall i \in [\rho],\ z_{i,e_i} := \mathsf{Open}_k(c^z_{i,e_i}, d^z_{i,e_i})$

outputs x if :

$\forall i \in [\rho],\ \mathsf{Verify}(x, a_i, e_i, z_{i,e_i}) = 1$

Simulating Corrupt Provers

1^κ

k

$(k, t) \leftarrow \text{TGen}(1^\kappa)$

x

$\forall i \in [\rho],\ e_i \leftarrow E_x$

$\forall i \in [\rho],\ (c_i^e, d_i^e) \leftarrow \text{Com}_k(e_i)$

$c_1^e, \ldots, c_\rho^e$

$a_1, \ldots, a_\rho, \left\{ c^z_{1,\hat{e}}, \ldots, c^z_{\rho,\hat{e}} \right\}_{\hat{e} \in E_x}$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ z_{i,\hat{e}} := \text{Ext}_t(c^z_{i,\hat{e}})$

$\forall i \in [\rho],\ w_i := \text{Reconstruct}(x, a_i, \{\hat{e}, z_{i,\hat{e}}\}_{\hat{e} \in E_x})$

$w := w_i \ \text{ s.t. } \ w_i \neq \perp$

$d_1^e, \ldots, d_\rho^e$

$d^z_{1,e_1}, \ldots, d^z_{\rho,e_\rho}$

$\forall i \in [\rho],\ z'_{i,e_i} := \text{Open}_k(c^z_{i,e_i}, d^z_{i,e_i})$

sends (x, w) to $\mathcal{F}^R_{\text{ZK}}$ if :

$\forall i \in [\rho],\ \text{Verify}(x, a_i, e_i, z'_{i,e_i}) = 1$

rupt Provers

$(k, t) \leftarrow \mathsf{TGen}(1^\kappa)$

$\forall i \in [\rho],\ e_i \leftarrow E_x$

$\forall i \in [\rho],\ (c^e_i, d^e_i) \leftarrow \mathsf{Com}_k(e_i)$

$\left\{ c^z_{\rho,\hat{e}} \right\}_{\hat{e} \in E_x}$

$\forall i \in [\rho],\ \forall \hat{e} \in E_x,\ z_{i,\hat{e}} := \mathsf{Ext}_t(c^z_{i,\hat{e}})$

$\forall i \in [\rho],\ w_i := \mathsf{Reconstruct}(x, a_i, \{\hat{e}, z_{i,\hat{e}}\}_{\hat{e} \in E_x})$

$w := w_i \ \ \mathrm{s.t.} \ \ w_i \neq \perp$

$\forall i \in [\rho],\ z'_{i,e_i} := \mathsf{Open}_k(c^z_{i,e_i}, d^z_{i,e_i})$

sends (x, w) to $\mathcal{F}^R_{\mathsf{ZK}}$ if :

$\forall i \in [\rho],\ \mathsf{Verify}(x, a_i, e_i, z'_{i,e_i}) = 1$

In order to prove that this simulation is indistinguishable from the real world, we use one hybrid experiments.

In the hybrid, we generate k within $\mathcal{F}_{\mathsf{CRS}}$ using $(k, t) \leftarrow \mathsf{TGen}(1^\kappa)$ instead of $k \leftarrow \mathsf{Gen}(1^\kappa)$. This is indistinguishable from the real world by reduction to the extractable security of the commitment scheme.

In the ideal world, we extract $z_{i,\hat{e}} := \mathsf{Ext}_t(c^z_{i,\hat{e}})\ \forall i \in [\rho],\ \forall \hat{e} \in E_x$ and attempt $w_i := \mathsf{Reconstruct}(x, a_i, \{\hat{e}, z_{i,\hat{e}}\}_{\hat{e} \in E_x})\ \forall i \in [\rho]$. If $w_i = \perp\ \forall i \in [\rho]$, we silently halt.

The ideal world can be distinguished from the first hybrid *only* if the prover cheats by committing at least one invalid response in all ρ repetitions of the sigma protocol, and goes undetected in all of them, causing the verifier in the hybrid to output x while the verifier in the ideal world outputs nothing. This event happens with probability $\leq (1 - 1/\ell(|x|))^\rho = (1 - 1/\ell(|x|))^{\ell(|x|)\cdot\kappa} \leq e^{-\kappa}$, which is negligible.

Thus, there is a simulator such that the real and ideal worlds are indistinguishable when either party is corrupt, and we have the theorem that we wished for. ∎

Applying the Protocol Compiler

Theorem 3: Let ℓ be a polynomial and let L be a language with membership-verifier algorithm R. If there exists an $\ell(|x|)$-special-sound SHVZK sigma protocol for L, and there exists an *extractable* commitment scheme, then there exists a protocol in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model that realizes $\mathcal{F}_{\mathsf{ZK}}^{R}$.

Recall **Theorem 1':** Assuming perfectly-binding commitment schemes exist, there is an $|E|$-special-sound SHVZK sigma protocol for L_{G3C}, where $|E|$ is the number of edges in the instance.

Recall **Theorem 2':** deciding an L_{CSAT} instance C can be reduced in deterministic polynomial time to deciding an L_{G3C} instance $G = (V, E)$ where $|E| \in O(|C|)$.

Corollary 1: For every NP-language L with membership-verifier algorithm R, there exists a protocol in the CRS model that realizes $\mathcal{F}_{\mathsf{ZK}}^{R}$, assuming the existence of extractable, perfectly binding commitments.

Proof: By the above theorems, plus Cook-Levin. ∎

Corollary 2: For every NP-language L with membership-verifier algorithm R, there exists a protocol in the CRS model that realizes $\mathcal{F}_{\mathsf{ZK}}^{R}$, assuming the Decisional Diffie-Hellman Assumption is true.

Proof: By Corollary 1, plus ElGamal commitments. ∎

Equivocable and Extractable Commitments

Now we will put $\mathcal{F}_{\mathsf{ZK}}$ to use! For some commitment scheme $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ on message space $\mathcal{M}$, let R_{Com} be a membership verifier for the image of Com_k; i.e. $R_{\mathsf{Com}}((k,c),(m,r)) = 1 \iff \mathsf{Com}_k(m;r) = c$.

Also, let R_{Open} be a verifier for the image of $\mathsf{Com}_k(m)$; i.e. $R_{\mathsf{open}}((k,c,m),r) = 1 \iff \mathsf{Com}_k(m;r) = c$.

Theorem 4: If there exists a perfectly binding commitment scheme, then there is a commitment scheme in the $(\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Com}}}, \mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Open}}})$-hybrid model that is extractable, equivocable, *and* perfectly binding.

Proof Sketch: The CRS consists of $k \leftarrow \mathsf{Gen}(1^\kappa)$.

To commit a message $m \in \mathcal{M}$, a sender samples r from the appropriate set, computes $c := \mathsf{Com}_k(m;r)$, sends c to the receiver. Then the sender sends $((k,c),(m,r))$ to $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Com}}}$ and the receiver checks that the values (k,c) it receives from $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Com}}}$, match the ones the sender transmitted directly. The decommitment is (m,r). In the ideal world, $\mathcal{S}$ can extract m via $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Com}}}$.

To open a commitment c under key k using the decommitment (m,r), the sender transmits m to the receiver and sends $((k,c,m),r)$ to $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Open}}}$. The receiver checks that the values (k,c,m) it receives match the ones the sender transmitted directly. In the ideal world, $\mathcal{S}$ can equivocate by simply outputting any $m \in \mathcal{M}$ on behalf of $\mathcal{F}_{\mathsf{ZK}}^{R_{\mathsf{Open}}}$. m will not correspond to c, but computational hiding guarantees that this fact cannot be used to distinguish. ∎

CS4501 Cryptographic Protocols
Lecture 25: ZK for all of NP, Proofs of Knowledge, SLE

https://jackdoerner.net/teaching/#2026/Spring/CS4501