Perfect Zero-Knowledge Proof Systems

Does the verifier learn anything from the interaction?

What does it mean to learn something?

Perfect Zero-Knowledge Proof Systems

Does the verifier learn anything from the interaction? What does it mean to learn something?

Idea: the verifier learns something if, after the interaction, he is able to compute some function that he was not able to compute before the interaction

How do we formalize this?

Perfect Zero-Knowledge Proof Systems

Does the verifier learn anything from the interaction? What does it mean to learn something?

Idea: the verifier learns something if, after the interaction, he is able to compute some function that he was not able to compute before the interaction

How do we formalize this?

Let $\text{view}_V^P(x)$ be a random variable denoting the transcript of the message received by V during an interaction between P and V with common input x, plus all the random bits used by V.

Perfect Zero-Knowledge Proof Systems

Does the verifier learn anything from the interaction? What does it mean to learn something?

Idea: the verifier learns something if, after the interaction, he is able to compute some function that he was not able to compute before the interaction

How do we formalize this?

Let $\text{view}_V^P(x)$ be a random variable denoting the transcript of the message received by V during an interaction between P and V with common input x, plus all the random bits used by V.

Definition: An interactive proof system (P, V) for a language L is **perfect zero-knowledge** if for every probabilistic polynomial-time verifier V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are identically distributed for every $x \in L$:

- $\text{view}_{V^*}^P(x)$
- $M^*(x)$

Perfect Zero-Knowledge Proof Systems

Does the verifier learn anything from the interaction? What does it mean to learn something?

Idea: the verifier learns something if, after the interaction, he is able to compute some function that he was not able to compute before the interaction

How do we formalize this?

Let $\text{view}_V^P(x)$ be a random variable denoting the transcript of the message received by V during an interaction between P and V with common input x, plus all the random bits used by V.

Definition: An interactive proof system (P, V) for a language L is **perfect zero-knowledge** if for every probabilistic polynomial-time verifier V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are identically distributed for every $x \in L$:

- $\text{view}_{V^*}^P(x)$
- $M^*(x)$

The algorithm M^* is called a **simulator** for the interaction of V^* with P.

Note: the condition must hold **for every** verifier V^*
(even one that deviates from the protocol and tries to trick the prover into leaking information)

The Simulation Paradigm

If anything that the verifier can compute after the interaction can also be computed without the interaction, then the verifier gained no information from the interaction

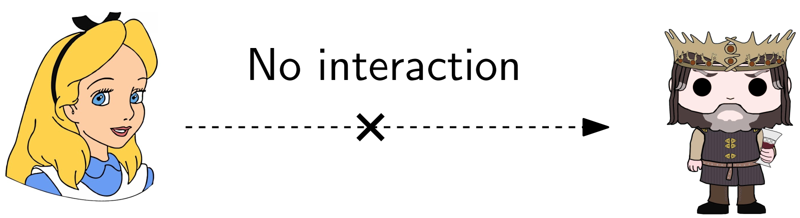
If M^* is a simulator for V^*, then the execution of an algorithm $A(x, \text{view}_P^{V^*})$ can always be simulated by an algorithm that returns $A(x, M^*(x))$

The Simulation Paradigm

If anything that the verifier can compute after the interaction can also be computed without the interaction, then the verifier gained no information from the interaction

If M^* is a simulator for V^*, then the execution of an algorithm $A(x, \text{view}_P^{V^*})$ can always be simulated by an algorithm that returns $A(x, M^*(x))$

It is trivial to design a perfect zero-knowledge proof system for any language in BPP.



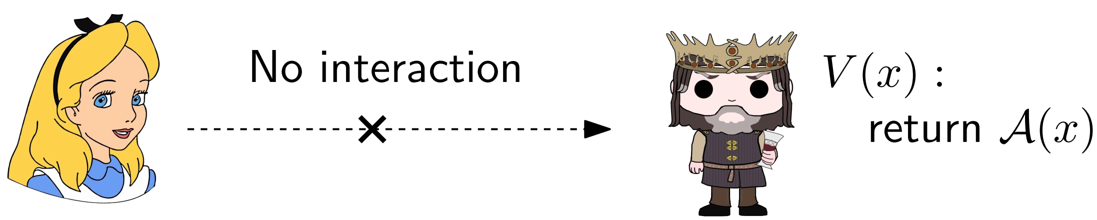
$V(x)$:
  return $\mathcal{A}(x)$

The Simulation Paradigm

If anything that the verifier can compute after the interaction can also be computed without the interaction, then the verifier gained no information from the interaction

If M^* is a simulator for V^*, then the execution of an algorithm $A(x, \text{view}_P^{V^*})$ can always be simulated by an algorithm that returns $A(x, M^*(x))$

It is trivial to design a perfect zero-knowledge proof system for any language in BPP.



Simulator: Take an interactive protocol (P, V) in which P does not interact with V and, given V^*, define M^* as follows:

- Simulate an execution of $V^*(x)$ and record the value of all random bits used during the execution
- Return the recorded view

Relaxing the Simulator Requirements

We can relax the requirements on the simulator

For every V^*, it suffices to have a probabilistic polynomial-time algorithm $\widetilde{M}^P_{V^*}$ such that:

- $\widetilde{M}^P_{V^*}(x)$ succeeds with probability at least $\frac{1}{p(n)}$ and returns a (simulated) view, or it fails
- Let $m(x)$ be the random variable describing the output of $\widetilde{M}^P_{V^*}(x)$ conditioned on the event that the execution succeeds. The variables $m(x)$ and $\text{view}^P_{V^*}(x)$ are identically distributed

Relaxing the Simulator Requirements

We can relax the requirements on the simulator

For every V^*, it suffices to have a probabilistic polynomial-time algorithm $\widetilde{M}^P_{V^*}$ such that:

- $\widetilde{M}^P_{V^*}(x)$ succeeds with probability at least $\frac{1}{p(n)}$ and returns a (simulated) view, or it fails
- Let $m(x)$ be the random variable describing the output of $\widetilde{M}^P_{V^*}(x)$ conditioned on the event that the execution succeeds. The variables $m(x)$ and $\text{view}^P_{V^*}(x)$ are identically distributed

The existence of $\widetilde{M}^P_{V^*}$ implies the existence of a simulator $M^P_{V^*}$ that runs in expected polynomial-time:

Simulator $M^P_{V^*}(x)$:

- $v \leftarrow \perp$
- While $v \neq \perp$:
  - $v \leftarrow \widetilde{M}^P_{V^*}(x)$
- Return v

Relaxing the Simulator Requirements

We can relax the requirements on the simulator

For every V^*, it suffices to have a probabilistic polynomial-time algorithm $\widetilde{M}_{V^*}^P$ such that:

- $\widetilde{M}_{V^*}^P(x)$ succeeds with probability at least $\frac{1}{p(n)}$ and returns a (simulated) view, or it fails
- Let $m(x)$ be the random variable describing the output of $\widetilde{M}_{V^*}^P(x)$ conditioned on the event that the execution succeeds. The variables $m(x)$ and $\text{view}_{V^*}^P(x)$ are identically distributed

The existence of $\widetilde{M}_{V^*}^P$ implies the existence of a simulator $M_{V^*}^P$ that runs in expected polynomial-time:

Simulator $M_{V^*}^P(x)$:

- $v \leftarrow \perp$
- While $v \neq \perp$:
  - $v \leftarrow \widetilde{M}_{V^*}^P(x)$
- Return v

The probability that $M_{V^*}^P$ does not halt in the first $k \cdot p(n)$ iterations is at most:

$$(1 - \frac{1}{p(n)})^{k \cdot p(n)} \leq e^{-k}$$

Relaxing the Simulator Requirements

We can relax the requirements on the simulator

For every V^*, it suffices to have a probabilistic polynomial-time algorithm $\widetilde{M}^P_{V^*}$ such that:

- $\widetilde{M}^P_{V^*}(x)$ succeeds with probability at least $\frac{1}{p(n)}$ and returns a (simulated) view, or it fails
- Let $m(x)$ be the random variable describing the output of $\widetilde{M}^P_{V^*}(x)$ conditioned on the event that the execution succeeds. The variables $m(x)$ and $\text{view}^P_{V^*}(x)$ are identically distributed

The existence of $\widetilde{M}^P_{V^*}$ implies the existence of a simulator $M^P_{V^*}$ that runs in expected polynomial-time:

Simulator $M^P_{V^*}(x)$:

- $v \leftarrow \bot$
- While $v \neq \bot$:
  - $v \leftarrow \widetilde{M}^P_{V^*}(x)$
- Return v

The probability that $M^P_{V^*}$ does not halt in the first $k \cdot p(n)$ iterations is at most:

$$(1 - \tfrac{1}{p(n)})^{k \cdot p(n)} \leq e^{-k}$$

Let T be the number of "groups" of $p(n)$ iterations that are (completely or partially) executed

$$\mathbb{E}[T] = \textstyle\sum_{k=1}^{\infty} \Pr[T \geq k] \ \leq \sum_{k=0}^{\infty} e^{-k}$$

Relaxing the Simulator Requirements

We can relax the requirements on the simulator

For every V^*, it suffices to have a probabilistic polynomial-time algorithm $\widetilde{M}^P_{V^*}$ such that:

- $\widetilde{M}^P_{V^*}(x)$ succeeds with probability at least $\frac{1}{p(n)}$ and returns a (simulated) view, or it fails
- Let $m(x)$ be the random variable describing the output of $\widetilde{M}^P_{V^*}(x)$ conditioned on the event that the execution succeeds. The variables $m(x)$ and $\text{view}^P_{V^*}(x)$ are identically distributed

The existence of $\widetilde{M}^P_{V^*}$ implies the existence of a simulator $M^P_{V^*}$ that runs in expected polynomial-time:

Simulator $M^P_{V^*}(x)$:

- $v \leftarrow \perp$
- While $v \neq \perp$:
  - $v \leftarrow \widetilde{M}^P_{V^*}(x)$
- Return v

The probability that $M^P_{V^*}$ does not halt in the first $k \cdot p(n)$ iterations is at most:

$$(1 - \tfrac{1}{p(n)})^{k \cdot p(n)} \leq e^{-k}$$

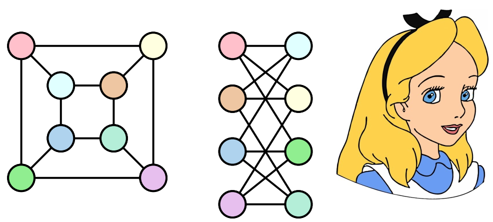
Let T be the number of “groups” of $p(n)$ iterations that are (completely or partially) executed

$$\mathbb{E}[T] = \textstyle\sum_{k=1}^{\infty} \Pr[T \geq k] \ \leq \sum_{k=0}^{\infty} e^{-k} \quad = \frac{1}{1-e^{-1}} \quad < 2$$

A PZK Proof System for Graph Isomorphism

The language L contains all pairs of graphs (G_1, G_2) such that $G_2 = \phi(G_1)$ for some permutation $\phi : V \to V$

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

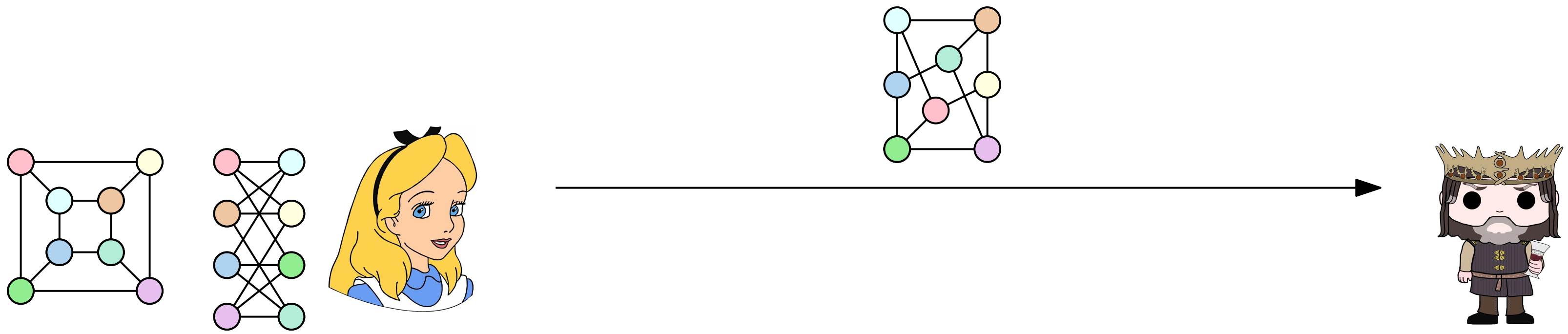


A PZK Proof System for Graph Isomorphism

The language L contains all pairs of graphs (G_1, G_2) such that $G_2 = \phi(G_1)$ for some permutation $\phi : V \to V$

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The prover picks a random permutation $\pi : V \to V$ and sends the graph $G' = \pi(G_2)$ to the verifier

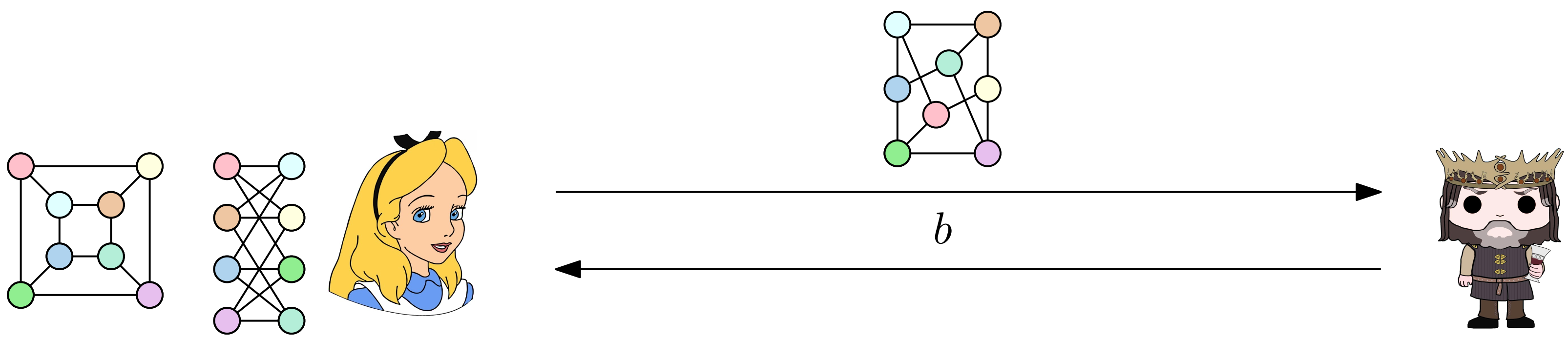


A PZK Proof System for Graph Isomorphism

The language L contains all pairs of graphs (G_1, G_2) such that $G_2 = \phi(G_1)$ for some permutation $\phi : V \to V$

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The prover picks a random permutation $\pi : V \to V$ and sends the graph $G' = \pi(G_2)$ to the verifier
- The verifier chooses b u.a.r. in $\{1, 2\}$ and sends b to the prover



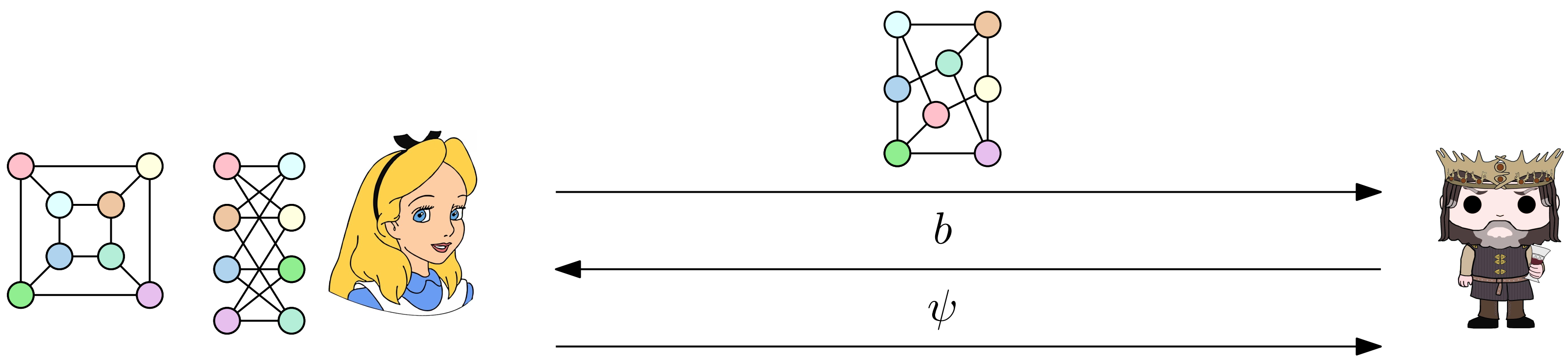
A PZK Proof System for Graph Isomorphism

The language L contains all pairs of graphs (G_1, G_2) such that $G_2 = \phi(G_1)$ for some permutation $\phi : V \to V$

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The prover picks a random permutation $\pi : V \to V$ and sends the graph $G' = \pi(G_2)$ to the verifier
- The verifier chooses b u.a.r. in $\{1, 2\}$ and sends b to the prover
- If $b = 2$, the prover replies with $\psi = \pi$.

  Otherwise the prover replies with $\psi = \pi \circ \phi$ (i.e., $\psi(v) = \pi(\phi(v))$)



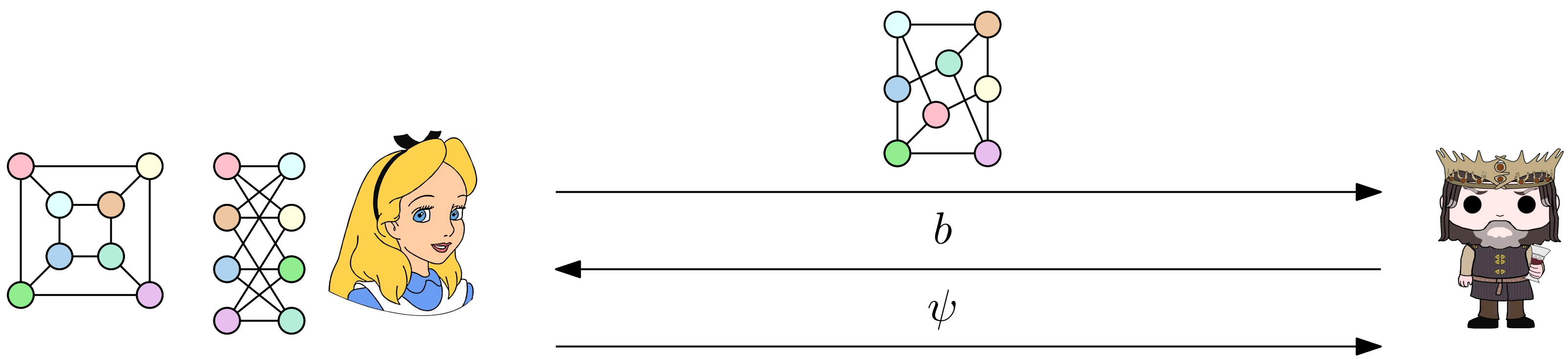
A PZK Proof System for Graph Isomorphism

The language L contains all pairs of graphs (G_1, G_2) such that $G_2 = \phi(G_1)$ for some permutation $\phi : V \to V$

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The prover picks a random permutation $\pi : V \to V$ and sends the graph $G' = \pi(G_2)$ to the verifier
- The verifier chooses b u.a.r. in $\{1, 2\}$ and sends b to the prover
- If $b = 2$, the prover replies with $\psi = \pi$.

  Otherwise the prover replies with $\psi = \pi \circ \phi$ (i.e., $\psi(v) = \pi(\phi(v))$
- The verifier accepts iff $\psi(G_b) = G'$



A PZK Proof System for Graph Isomorphism

Completeness:

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

A PZK Proof System for Graph Isomorphism

Completeness:

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2)$

A PZK Proof System for Graph Isomorphism

Completeness:

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \implies$ the verifier accepts

A PZK Proof System for Graph Isomorphism

Completeness:

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \quad \implies$ the verifier accepts

If $b = 1$, the prover replies with $\psi = \pi \circ \phi$. $G' = \pi(G_2) = \pi(\phi(G_1)) = \psi(G_1)$

A PZK Proof System for Graph Isomorphism

Completeness: $\Pr[V \text{ accepts}] = 1$

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \implies$ the verifier accepts

If $b = 1$, the prover replies with $\psi = \pi \circ \phi$. $G' = \pi(G_2) = \pi(\phi(G_1)) = \psi(G_1) \implies$ the verifier accepts

A PZK Proof System for Graph Isomorphism

Completeness: $\Pr[V \text{ accepts}] = 1$

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \implies$ the verifier accepts

If $b = 1$, the prover replies with $\psi = \pi \circ \phi$. $G' = \pi(G_2) = \pi(\phi(G_1)) = \psi(G_1) \implies$ the verifier accepts

Soundness:

Consider $(G_1, G_2) \notin L$, i.e., there is no isomorphism between G_1 and G_2

The graph G' sent by the prover is isomorphic to at most one of G_1 and G_2

A PZK Proof System for Graph Isomorphism

Completeness: $\Pr[V \text{ accepts}] = 1$

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \implies$ the verifier accepts

If $b = 1$, the prover replies with $\psi = \pi \circ \phi$. $G' = \pi(G_2) = \pi(\phi(G_1)) = \psi(G_1) \implies$ the verifier accepts

Soundness: $\Pr[V \text{ accepts}] \leq \frac{1}{2}$

Consider $(G_1, G_2) \notin L$, i.e., there is no isomorphism between G_1 and G_2

The graph G' sent by the prover is isomorphic to at most one of G_1 and G_2

Since the verifier chooses b u.a.r. in $\{1, 2\}$, there is a probability of at least $\frac{1}{2}$ that G_b is not isomorphic to G'

In this case, the verifier cannot reply with an isomorphism ψ that satisfies $\psi(G_b) = G'$

A PZK Proof System for Graph Isomorphism

Completeness: $\Pr[V \text{ accepts}] = 1$

Consider $(G_1, G_2) \in L$, i.e., there is an isomorphism ϕ between G_1 and G_2

If $b = 2$, the prover replies with $\psi = \pi$. $G' = \pi(G_2) = \psi(G_2) \implies$ the verifier accepts

If $b = 1$, the prover replies with $\psi = \pi \circ \phi$. $G' = \pi(G_2) = \pi(\phi(G_1)) = \psi(G_1) \implies$ the verifier accepts

Soundness: $\Pr[V \text{ accepts}] \leq \frac{1}{2}$

Consider $(G_1, G_2) \notin L$, i.e., there is no isomorphism between G_1 and G_2

The graph G' sent by the prover is isomorphic to at most one of G_1 and G_2

Since the verifier chooses b u.a.r. in $\{1, 2\}$, there is a probability of at least $\frac{1}{2}$ that G_b is not isomorphic to G'

In this case, the verifier cannot reply with an isomorphism ψ that satisfies $\psi(G_b) = G'$

Use probability amplification

A PZK Proof System for Graph Isomorphism

Zero Knowledge:

For any $(G_1, G_2) \in L$ and any (possibly cheating) verifier V^*, we need to come up with a simulator $M_{V^*}^P$ for the interaction between P and V^*

A PZK Proof System for Graph Isomorphism

Zero Knowledge:

For any $(G_1, G_2) \in L$ and any (possibly cheating) verifier V^*, we need to come up with a simulator $M^P_{V^*}$ for the interaction between P and V^*

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M^P_V(x)$:

- Choose $\widetilde{b}$ u.a.r. in $\{1, 2\}$
- Pick a random permutation $\psi : V \to V$
- Pretend to "send" $\psi(G_{\widetilde{b}})$ from the prover to the verifier
- (The verifier "chooses" $b = \widetilde{b}$)
- Pretend to "send" ψ from the prover to the verifier
- The view consists of all the simulated messages sent from the prover to the verifier and of the value b

A PZK Proof System for Graph Isomorphism

Zero Knowledge:

For any $(G_1, G_2) \in L$ and any (possibly cheating) verifier V^*, we need to come up with a simulator $M^P_{V^*}$ for the interaction between P and V^*

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M^P_V(x)$:

- Choose $\widetilde{b}$ u.a.r. in $\{1, 2\}$
- Pick a random permutation $\psi : V \to V$
- Pretend to "send" $\psi(G_{\widetilde{b}})$ from the prover to the verifier
- (The verifier "chooses" $b = \widetilde{b}$)
- Pretend to "send" ψ from the prover to the verifier
- The view consists of all the simulated messages sent from the prover to the verifier and of the value b

How do we handle cheating verifiers?

A PZK Proof System for Graph Isomorphism

Idea: the only thing that a cheating verifier can do is to choose the value of b (as a function of G')

If the chosen value b happens to be $\widetilde{b}$, then the previous strategy works

A PZK Proof System for Graph Isomorphism

Idea: the only thing that a cheating verifier can do is to choose the value of b (as a function of G')

If the chosen value b happens to be $\widetilde{b}$, then the previous strategy works

Simulator $M^P_{V^*}(x)$**:**

- Choose $\widetilde{b}$ u.a.r. in $\{1, 2\}$
- Pick a random permutation $\psi : V \to V$
- Simulate the verifier $V^*(x)$ and send $G' = \psi(G_{\widetilde{b}})$ to it
- "Receive" the bit b that the simulated verifier sends to the prover

Assume that $b \in \{1, 2\}$
(can be handled)

A PZK Proof System for Graph Isomorphism

Idea: the only thing that a cheating verifier can do is to choose the value of b (as a function of G')

If the chosen value b happens to be $\widetilde{b}$, then the previous strategy works

Simulator $M^P_{V^*}(x)$**:**

- Choose $\widetilde{b}$ u.a.r. in $\{1,2\}$
- Pick a random permutation $\psi : V \to V$
- Simulate the verifier $V^*(x)$ and send $G' = \psi(G_{\widetilde{b}})$ to it
- "Receive" the bit b that the simulated verifier sends to the prover
- If $\widetilde{b} \neq b$: halt the simulator and **fail**

Assume that $b \in \{1,2\}$
(can be handled)

A PZK Proof System for Graph Isomorphism

Idea: the only thing that a cheating verifier can do is to choose the value of b (as a function of G')

If the chosen value b happens to be $\widetilde{b}$, then the previous strategy works

Simulator $M^P_{V^*}(x)$**:**

- Choose $\widetilde{b}$ u.a.r. in $\{1, 2\}$
- Pick a random permutation $\psi : V \to V$
- Simulate the verifier $V^*(x)$ and send $G' = \psi(G_{\widetilde{b}})$ to it
- "Receive" the bit b that the simulated verifier sends to the prover
- If $\widetilde{b} \neq b$: halt the simulator and **fail**
- Pretend to "send" ψ from the prover to the verifier, and continue the simulation of $V^*(x)$
- The view consists of all the simulated messages sent from the prover to the verifier and of all the random bits used by simulated execution of $V^*(x)$

Assume that $b \in \{1, 2\}$
(can be handled)

A PZK Proof System for Graph Isomorphism

Idea: the only thing that a cheating verifier can do is to choose the value of b (as a function of G')

If the chosen value b happens to be $\widetilde{b}$, then the previous strategy works

Simulator $M^P_{V^*}(x)$**:**

- Choose $\widetilde{b}$ u.a.r. in $\{1, 2\}$
- Pick a random permutation $\psi : V \to V$
- Simulate the verifier $V^*(x)$ and send $G' = \psi(G_{\widetilde{b}})$ to it
- "Receive" the bit b that the simulated verifier sends to the prover
- If $\widetilde{b} \neq b$: halt the simulator and **fail**
- Pretend to "send" ψ from the prover to the verifier, and continue the simulation of $V^*(x)$
- The view consists of all the simulated messages sent from the prover to the verifier and of all the random bits used by simulated execution of $V^*(x)$

Assume that $b \in \{1, 2\}$
(can be handled)

What's the failure probability?

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibily randomized) process used by the verifier to compute b from G' (and x)

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$\Pr[M_{V^*}^{P} \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\boxed{\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H}$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$$\Pr[M_{V^*}^{P} \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$$

$$\Pr[\widetilde{b} = 1 \wedge b = 2] = \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \frac{1}{2}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$\Pr[M^P_{V^*} \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$

$$\Pr[\widetilde{b} = 1 \wedge b = 2] = \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2}$$
$$= \tfrac{1}{2} \sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H]$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\boxed{\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H}$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$\Pr[M_{V^*}^{P} \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$

$$\begin{aligned}\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H]\end{aligned}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$\Pr[M_{V^*}^P \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$

$$\begin{aligned}
\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot p_H
\end{aligned}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$\Pr[M_{V^*}^P \text{ fails}] = \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1]$

$$\begin{aligned}
\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot p_H
\end{aligned}$$

$$\begin{aligned}
\Pr[\widetilde{b} = 2 \wedge b = 1] &= \Pr[b = 1 \mid \widetilde{b} = 2] \cdot \Pr[\widetilde{b} = 2] = \Pr[R(\psi(G_2)) = 1] \cdot \tfrac{1}{2} \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1 \wedge \psi(G_2) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot \Pr[\psi(G_2) = H] \\
&= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot p_H
\end{aligned}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$$\begin{aligned}\Pr[M^{P}_{V^*} \text{ fails}] &= \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1] \\ &= \tfrac{1}{2} \textstyle\sum_H \left(\Pr[R(G') = 2] \cdot p_H + \Pr[R(G') = 1] \cdot p_H\right)\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot p_H\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 2 \wedge b = 1] &= \Pr[b = 1 \mid \widetilde{b} = 2] \cdot \Pr[\widetilde{b} = 2] = \Pr[R(\psi(G_2)) = 1] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1 \wedge \psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot \Pr[\psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot p_H\end{aligned}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$$\begin{aligned}\Pr[M_{V^*}^P \text{ fails}] &= \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1] \\ &= \tfrac{1}{2} \textstyle\sum_H \left(\Pr[R(G') = 2] \cdot p_H + \Pr[R(G') = 1] \cdot p_H\right) = \tfrac{1}{2} \textstyle\sum_H p_H\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot p_H\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 2 \wedge b = 1] &= \Pr[b = 1 \mid \widetilde{b} = 2] \cdot \Pr[\widetilde{b} = 2] = \Pr[R(\psi(G_2)) = 1] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1 \wedge \psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot \Pr[\psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot p_H\end{aligned}$$

A PZK Proof System for Graph Isomorphism

Since G_1 and G_2 are isomorphic: $\Pr[\psi(G_1) = H] = \Pr[\psi(G_2) = H] = p_H$

Let $R(G')$ be the (possibly randomized) process used by the verifier to compute b from G' (and x)

$$\begin{aligned}\Pr[M_{V^*}^{P} \text{ fails}] &= \Pr[\widetilde{b} = 1 \wedge b = 2] + \Pr[\widetilde{b} = 2 \wedge b = 1] \\ &= \tfrac{1}{2} \textstyle\sum_H \left(\Pr[R(G') = 2] \cdot p_H + \Pr[R(G') = 1] \cdot p_H\right) = \tfrac{1}{2} \textstyle\sum_H p_H = \tfrac{1}{2}\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 1 \wedge b = 2] &= \Pr[b = 2 \mid \widetilde{b} = 1] \cdot \Pr[\widetilde{b} = 1] = \Pr[R(\psi(G_1)) = 2] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2 \wedge \psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot \Pr[\psi(G_1) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 2] \cdot p_H\end{aligned}$$

$$\begin{aligned}\Pr[\widetilde{b} = 2 \wedge b = 1] &= \Pr[b = 1 \mid \widetilde{b} = 2] \cdot \Pr[\widetilde{b} = 2] = \Pr[R(\psi(G_2)) = 1] \cdot \tfrac{1}{2} \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1 \wedge \psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot \Pr[\psi(G_2) = H] \\ &= \tfrac{1}{2} \textstyle\sum_H \Pr[R(H) = 1] \cdot p_H\end{aligned}$$

A PZK Proof System for Graph Isomorphism

If G_1 and G_2 are isomorphic then, after interacting with the prover, the verifier will be convinced that:

- There **exists** an isomorphism ψ between G_1 and G_2 (i.e., there exists a witness for $x \in L$)
- The prover **must know** ϕ (i.e., the prover **knows** a witness w for x)

A PZK Proof System for Graph Isomorphism

If G_1 and G_2 are isomorphic then, after interacting with the prover, the verifier will be convinced that:

- There **exists** an isomorphism ψ between G_1 and G_2 (i.e., there exists a witness for $x \in L$)
- The prover **must know** ϕ (i.e., the prover **knows** a witness w for x)

In general, a Zero-Knowledge proof system for a language $L \in$ NP only needs to convince the verifier of the **existence** of a witness

If the proof additionally convinces the verifier that the prover must **know** a witness, then it is called a **zero-knowledge proof-of-knowledge**

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in \mathsf{NP}$ admits an interactive proof system (and it is trivial)

Does every language $L \in \mathsf{NP}$ also admit an interactive **zero-knowledge** proof system?

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in \mathsf{NP}$ admits an interactive proof system (and it is trivial)

Does every language $L \in \mathsf{NP}$ also admit an interactive **zero-knowledge** proof system?

- If PRGs exists then yes, for a notion of zero-knowledge known as **computational zero-knowledge**
- The proof system is no longer trivial

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in$ NP admits an interactive proof system (and it is trivial)

Does every language $L \in$ NP also admit an interactive **zero-knowledge** proof system?

- If PRGs exists then yes, for a notion of zero-knowledge known as **computational zero-knowledge**
- The proof system is no longer trivial

Roadmap:

- Define **computational zero-knowledge** (CZK)

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in$ NP admits an interactive proof system (and it is trivial)

Does every language $L \in$ NP also admit an interactive **zero-knowledge** proof system?

- If PRGs exists then yes, for a notion of zero-knowledge known as **computational zero-knowledge**
- The proof system is no longer trivial

Roadmap:

- Define **computational zero-knowledge** (CZK)
- Design a CZK proof system for the graph 3-coloring language G3C, which is NP-complete
  - We will make use of a **commitment scheme**

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in \text{NP}$ admits an interactive proof system (and it is trivial)

Does every language $L \in \text{NP}$ also admit an interactive **zero-knowledge** proof system?

- If PRGs exists then yes, for a notion of zero-knowledge known as **computational zero-knowledge**
- The proof system is no longer trivial

Roadmap:

- Define **computational zero-knowledge** (CZK)
- Design a CZK proof system for the graph 3-coloring language G3C, which is NP-complete
  - We will make use of a **commitment scheme**
- Since $L \in \text{NP}$, $L \leq_p \text{G3C}$ and hence there is a polynomial-time reduction f from L to G3C

$$x \in L \iff f(x) \in \text{G3C}$$

Zero-Knowledge Proof Systems for all NP

We know that any language $L \in \mathsf{NP}$ admits an interactive proof system (and it is trivial)

Does every language $L \in \mathsf{NP}$ also admit an interactive **zero-knowledge** proof system?

- If PRGs exists then yes, for a notion of zero-knowledge known as **computational zero-knowledge**
- The proof system is no longer trivial

Roadmap:

- Define **computational zero-knowledge** (CZK)
- Design a CZK proof system for the graph 3-coloring language G3C, which is NP-complete
  - We will make use of a **commitment scheme**
- Since $L \in \mathsf{NP}$, $L \leq_p$ G3C and hence there is a polynomial-time reduction f from L to G3C

$$x \in L \iff f(x) \in \mathsf{G3C}$$

- To obtain a CZK proof system for L, the prover and the verifier can first reduce the instance x of L to an instance $x' = f(x)$ of G3C and then use the CZK proof for G3C

Computational Zero-Knowledge

Idea: If we consider a computation to be feasible if it can be performed in polynomial time, then we do not need to perfectly simulate the view of V^*

It suffices for the output distribution of the simulator $M^*(x)$ to be **computationally indistinguishable** from $\text{view}_{V^*}^{P}$

Computational Zero-Knowledge

Idea: If we consider a computation to be feasible if it can be performed in polynomial time, then we do not need to perfectly simulate the view of V^*

It suffices for the output distribution of the simulator $M^*(x)$ to be **computationally indistinguishable** from $\text{view}_{V^*}^{P}$

Intuition: Whatever can be **efficiently** computed after interacting with P on input $x \in L$ can also be **efficiently** computed from x (without any interaction)

Computational Zero-Knowledge

Idea: If we consider a computation to be feasible if it can be performed in polynomial time, then we do not need to perfectly simulate the view of V^*

It suffices for the output distribution of the simulator $M^*(x)$ to be **computationally indistinguishable** from view$_{V^*}^{P}$

Intuition: Whatever can be **efficiently** computed after interacting with P on input $x \in L$ can also be **efficiently** computed from x (without any interaction)

Definition: An interactive proof system (P, V) for a language L is **computational zero-knowledge** if for every probabilistic polynomial-time verifier V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are **computationally indistinguishable** for every $x \in L$:

- view$_{V^*}^{P}(x)$
- $M^*(x)$

Computational Zero-Knowledge

Idea: If we consider a computation to be feasible if it can be performed in polynomial time, then we do not need to perfectly simulate the view of V^*

It suffices for the output distribution of the simulator $M^*(x)$ to be **computationally indistinguishable** from view$_{V^*}^{P}$

Intuition: Whatever can be **efficiently** computed after interacting with P on input $x \in L$ can also be **efficiently** computed from x (without any interaction)

Definition: An interactive proof system (P, V) for a language L is **computational zero-knowledge** if for every probabilistic polynomial-time verifier V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are **computationally indistinguishable** for every $x \in L$:

- view$_{V^*}^{P}(x)$
- $M^*(x)$

We can also allow the simulator to fail as in the case of perfect zero-knowledge

Computational Zero-Knowledge

Idea: If we consider a computation to be feasible if it can be performed in polynomial time, then we do not need to perfectly simulate the view of V^*

It suffices for the output distribution of the simulator $M^*(x)$ to be **computationally indistinguishable** from view$^P_{V^*}$

Intuition: Whatever can be **efficiently** computed after interacting with P on input $x \in L$ can also be **efficiently** computed from x (without any interaction)

Definition: An interactive proof system (P, V) for a language L is **computational zero-knowledge** if for every probabilistic polynomial-time verifier V^* there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are **computationally indistinguishable** for every $x \in L$:

- view$^P_{V^*}(x)$
- $M^*(x)$

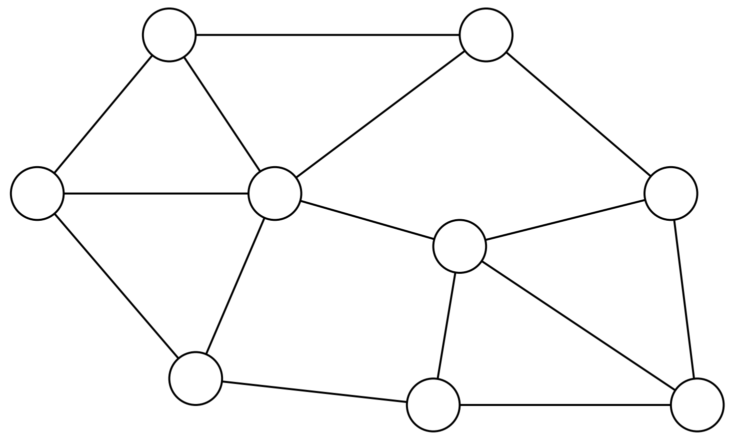
We can also allow the simulator to fail as in the case of perfect zero-knowledge

We define CZK (resp. PZK) as the class of all languages that admit a computational (resp. perfect) zero-knowledge proof system

$$\text{BPP} \subseteq \text{PZK} \subseteq \text{CZK} \subseteq \text{IP}$$

Graph 3-Coloring

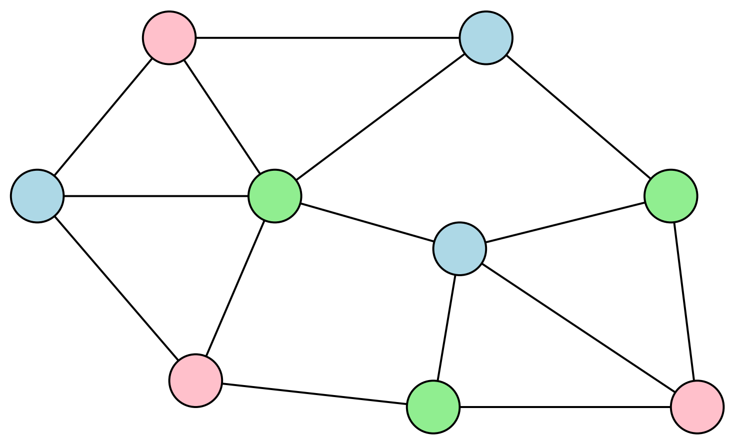
Let $G = (V, E)$ be a graph



A k-coloring of G is a function $c : V \rightarrow \{1, 2, \ldots, k\}$ such that each edge $(u, v) \in E$ has endpoints of the different "colors" (i.e., $c(u) \neq c(v)$)

Graph 3-Coloring

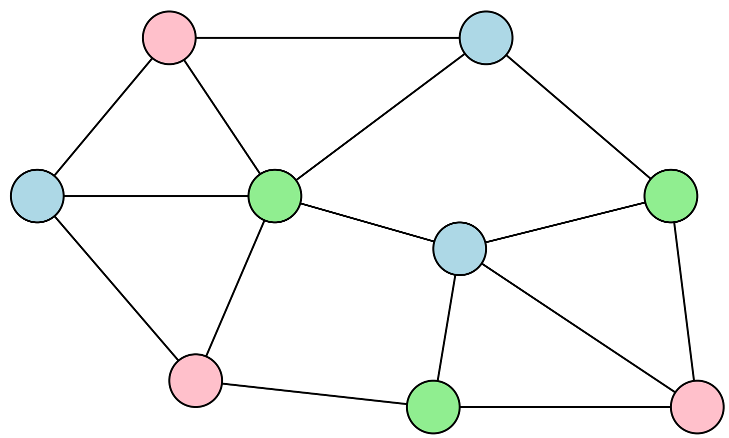
Let $G = (V, E)$ be a graph



A k-coloring of G is a function $c : V \to \{1, 2, \ldots, k\}$ such that each edge $(u, v) \in E$ has endpoints of the different "colors" (i.e., $c(u) \neq c(v)$)

Graph 3-Coloring

Let $G = (V, E)$ be a graph

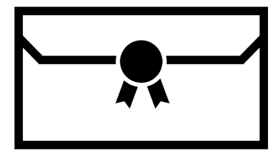


A k-coloring of G is a function $c : V \to \{1, 2, \ldots, k\}$ such that each edge $(u, v) \in E$ has endpoints of the different "colors" (i.e., $c(u) \neq c(v)$)

G3C is the set of all graphs G that admit a 3-coloring

Commitment schemes (Informal)

A commitment scheme is an interactive protocol that works in two phases, called **commit** and **reveal**, and allows a party, called **sender**, to:

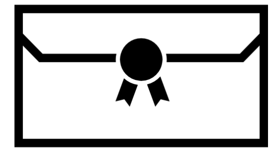


- Commit itself to a value m
- At a later time, "open" the commitment to reveal m

Commitment schemes (Informal)

A commitment scheme is an interactive protocol that works in two phases, called **commit** and **reveal**, and allows a party, called **sender**, to:

- Commit itself to a value m
- At a later time, "open" the commitment to reveal m



and satisfies the following properties:

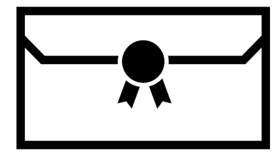
- **Hiding:** At the end of the commit phase, the other party, called the **receiver**, does not gain any information about the sender's value.

Even if the receiver cheats by deviating from the protocol!

Commitment schemes (Informal)

A commitment scheme is an interactive protocol that works in two phases, called **commit** and **reveal**, and allows a party, called **sender**, to:

- Commit itself to a value m
- At a later time, "open" the commitment to reveal m



and satisfies the following properties:

- **Hiding:** At the end of the commit phase, the other party, called the **receiver**, does not gain any information about the sender's value.

  Even if the receiver cheats by deviating from the protocol!

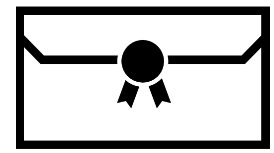
- **Binding:** Given the transcript of the interaction in the commit phase, there exists at most one value that the receiver can accept as a legal "opening" of the commitment

  Even if the sender cheats by deviating from the protocol!

Commitment schemes (Informal)

A commitment scheme is an interactive protocol that works in two phases, called **commit** and **reveal**, and allows a party, called **sender**, to:

- Commit itself to a value m
- At a later time, "open" the commitment to reveal m



and satisfies the following properties:

- **Hiding:** At the end of the commit phase, the other party, called the **receiver**, does not gain any information about the sender's value.

  Even if the receiver cheats by deviating from the protocol!

- **Binding:** Given the transcript of the interaction in the commit phase, there exists at most one value that the receiver can accept as a legal "opening" of the commitment

  Even if the sender cheats by deviating from the protocol!

- **Viability:** if both parties follow the protocol, then after the reveal phase, the receiver learns m

Commitment schemes (Formal)

A **bit-commitment scheme** is a pair of probabilistic polynomial-time interactive algorithms (S, R) where both S and R take 1^n as a common input, S takes a bit $b \in \{0, 1\}$ as a private input, and the following properties are satisfied:

Commitment schemes (Formal)

A **bit-commitment scheme** is a pair of probabilistic polynomial-time interactive algorithms (S, R) where both S and R take 1^n as a common input, S takes a bit $b \in \{0, 1\}$ as a private input, and the following properties are satisfied:

- **Hiding:** For every probabilistic polynomial-time algorithm R^*, the output of $(S(0), R^*)(1^n)$ is computationally indistinguishalbe from that of $(S(1), R^*)(1^n)$

Commitment schemes (Formal)

A **bit-commitment scheme** is a pair of probabilistic polynomial-time interactive algorithms (S, R) where both S and R take 1^n as a common input, S takes a bit $b \in \{0, 1\}$ as a private input, and the following properties are satisfied:

- **Hiding:** For every probabilistic polynomial-time algorithm R^*, the output of $(S(0), R^*)(1^n)$ is computationally indistinguishalbe from that of $(S(1), R^*)(1^n)$
- **Binding:**

  Let $(r, \overline{m})$ be the view of R, where r are the random coins used by R and $\overline{m}$ is a transcript of the messages received from the sender.

  For $\sigma \in \{0, 1\}$, we say that $(r, \overline{m})$ is a **possible** σ**-commitment** if it can be the view of R when S commits to σ. More precisely: there are some random bits s such that $\overline{m}$ are the messages received by $R(1^n)$ using random bits r in the interaction with $S(1^n, \sigma)$ using random bits s

Commitment schemes (Formal)

A **bit-commitment scheme** is a pair of probabilistic polynomial-time interactive algorithms (S, R) where both S and R take 1^n as a common input, S takes a bit $b \in \{0, 1\}$ as a private input, and the following properties are satisfied:

- **Hiding:** For every probabilistic polynomial-time algorithm R^*, the output of $(S(0), R^*)(1^n)$ is computationally indistinguishalbe from that of $(S(1), R^*)(1^n)$

- **Binding:**

  Let $(r, \overline{m})$ be the view of R, where r are the random coins used by R and $\overline{m}$ is a transcript of the messages received from the sender.

  For $\sigma \in \{0, 1\}$, we say that $(r, \overline{m})$ is a **possible σ-commitment** if it can be the view of R when S commits to σ. More precisely: there are some random bits s such that $\overline{m}$ are the messages received by $R(1^n)$ using random bits r in the interaction with $S(1^n, \sigma)$ using random bits s

  $(r, \overline{m})$ is **ambiguous** if its **both** a possible 0-commitment and a possible 1-commitment

  We require that for all but a negligible fraction of the strings $r \in \{0, 1\}^{\mathsf{poly}(n)}$, **there exists no $\overline{m}$** such that $(r, \overline{m})$ is ambiguous

Commitment schemes

The viability requirement is implicitly enforced by the formalization of the **binding** condition

There is a canonical way to **open** the commitment:

- The sender sends to the receiver the secret value b and sequence s of the random bits it used in the commitment phase
- The receiver simulates the interaction between itself (with random bits r) and $S(1^n, b)$ with random bits s and checks that the messages "sent" by the simulated S match those in $\overline{m}$

Commitment schemes

The viability requirement is implicitly enforced by the formalization of the **binding** condition

There is a canonical way to **open** the commitment:

- The sender sends to the receiver the secret value b and sequence s of the random bits it used in the commitment phase
- The receiver simulates the interaction between itself (with random bits r) and $S(1^n, b)$ with random bits s and checks that the messages "sent" by the simulated S match those in $\overline{m}$

Observation: if we have a bit-commitement scheme then we can build a commitment scheme for any binary string by committing each bit separately

Commitment schemes

The viability requirement is implicitly enforced by the formalization of the **binding** condition

There is a canonical way to **open** the commitment:

- The sender sends to the receiver the secret value b and sequence s of the random bits it used in the commitment phase
- The receiver simulates the interaction between itself (with random bits r) and $S(1^n, b)$ with random bits s and checks that the messages "sent" by the simulated S match those in $\overline{m}$

Observation: if we have a bit-commitement scheme then we can build a commitment scheme for any binary string by committing each bit separately

The above commitment schemes are:

- **Perfectly binding:** except that for a negligible probability, the sender is bound to the committed value (regardless of its computational power)

Commitment schemes

The viability requirement is implicitly enforced by the formalization of the **binding** condition

There is a canonical way to **open** the commitment:

- The sender sends to the receiver the secret value b and sequence s of the random bits it used in the commitment phase
- The receiver simulates the interaction between itself (with random bits r) and $S(1^n, b)$ with random bits s and checks that the messages "sent" by the simulated S match those in $\overline{m}$

Observation: if we have a bit-commitement scheme then we can build a commitment scheme for any binary string by committing each bit separately

The above commitment schemes are:

- **Perfectly binding:** except that for a negligible probability, the sender is bound to the committed value (regardless of its computational power)
- **Computationally hiding:** the committed value is hidden from a computationally bounded receiver. (A computationally unbounded receiver can bruteforce s)

(the commitment scheme based on hash functions does not suit our needs since it is not perfectly binding)

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Hiding:

Let U be a uniform random variable over $\{0,1\}^{3n}$ and let $\approx$ denote computational indistinguishability

For any $r \in \{0,1\}^{3n}$:

$$\underbrace{G(s) \approx U}_{G \text{ is a PRG}} \equiv \overbrace{U \oplus r}^{U \text{ is uniform}} \approx \underbrace{U \oplus r \approx G(s) \oplus r}_{G \text{ is a PRG}}$$

Therefore, $G(s)$ and $G(s) \oplus r$ are computationally indistinguishable

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Hiding:

Let U be a uniform random variable over $\{0,1\}^{3n}$ and let $\approx$ denote computational indistinguishability

For any $r \in \{0,1\}^{3n}$:

$$\underbrace{G(s) \approx U}_{G \text{ is a PRG}} \equiv \overbrace{U \oplus r}^{U \text{ is uniform}} \underbrace{ \approx G(s) \oplus r}_{G \text{ is a PRG}}$$

Therefore, $G(s)$ and $G(s) \oplus r$ are computationally indistinguishable

$\implies$ For any polynomial-time algorithm R^*, the output distributions of $R^*(1^n, G(s))$ and $R^*(1^n, G(s) \oplus r)$ are computationally indistinguishable

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Binding:

- We say that $r \in \{0,1\}^{3n}$ yields a *collision* if there are $s_1, s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r$
- If r does not yield a collision, then (r, α) is not ambiguous (regardless of α)

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Binding:

- We say that $r \in \{0,1\}^{3n}$ yields a *collision* if there are $s_1, s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r$
- If r does not yield a collision, then (r, α) is not ambiguous (regardless of α)
- For any pair s_1, s_2 there is exactly one string r that yields a collision (namely $r = G(s_1) \oplus G(s_2)$)

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

Binding:

- We say that $r \in \{0,1\}^{3n}$ yields a *collision* if there are $s_1, s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r$
- If r does not yield a collision, then (r, α) is not ambiguous (regardless of α)
- For any pair s_1, s_2 there is exactly one string r that yields a collision (namely $r = G(s_1) \oplus G(s_2)$)
- The number of strings r that yield a collision is at most $|\{0,1\}^n \times \{0,1\}^n| = 2^{2n}$
- The number possible strings r is 2^{3n}

Commitment schemes from PRGs

We can build a commitment scheme from a PRG $G : \{0,1\}^n \to \{0,1\}^{3n}$

Commit phase:

- The receiver chooses r u.a.r. from $\{0,1\}^{3n}$ and sends it to the sender
- The sender commits to b by selecting s u.a.r. from $\{0,1\}^n$ and sending $\alpha = G(s)$ if $b = 0$ and $\alpha = G(s) \oplus r$ otherwise

(Canonical) Reveal Phase:

- The sender reveals b and s
- The receiver accepts iff $b = 0$ and $G(s) = \alpha$, or $b = 1$ and $G(s) \oplus r = \alpha$

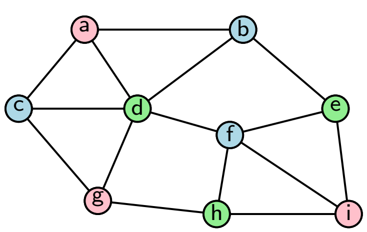
Binding:

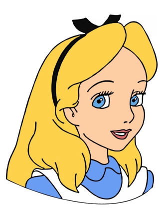
- We say that $r \in \{0,1\}^{3n}$ yields a *collision* if there are $s_1, s_2 \in \{0,1\}^n$ such that $G(s_1) = G(s_2) \oplus r$
- If r does not yield a collision, then (r, α) is not ambiguous (regardless of α)
- For any pair s_1, s_2 there is exactly one string r that yields a collision (namely $r = G(s_1) \oplus G(s_2)$)
- The number of strings r that yield a collision is at most $|\{0,1\}^n \times \{0,1\}^n| = 2^{2n}$
- The number possible strings r is 2^{3n}
- The fraction of strings r that can result in ambiguous (r, α) is at most $\frac{2^{2n}}{2^{3n}} = \frac{1}{2^n}$, which is negligible

A CZK Proof System for Graph 3-Coloring

Common input: $G = (V, E)$

The prover knows a 3-coloring c of G





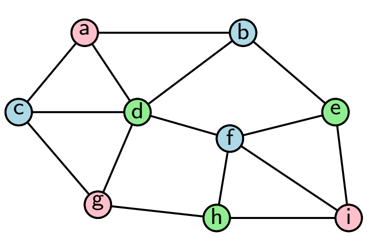

A CZK Proof System for Graph 3-Coloring

Common input: $G = (V, E)$

The prover knows a 3-coloring c of G

- The prover picks a random permutation π of $\{1, 2, 3\}$ and constructs the 3-coloring $c' = \pi \circ c$
- For each vertex v, the prover uses a commitment scheme to commit to $c'(v)$

  (commit to two bits for each vertex according to some encoding, e.g., 01 $\rightarrow 1$, 10 $\rightarrow 2$, $\{$00, 11$\} \rightarrow 3$)



A commitment to a color for $c'(z)$ every vertex z →

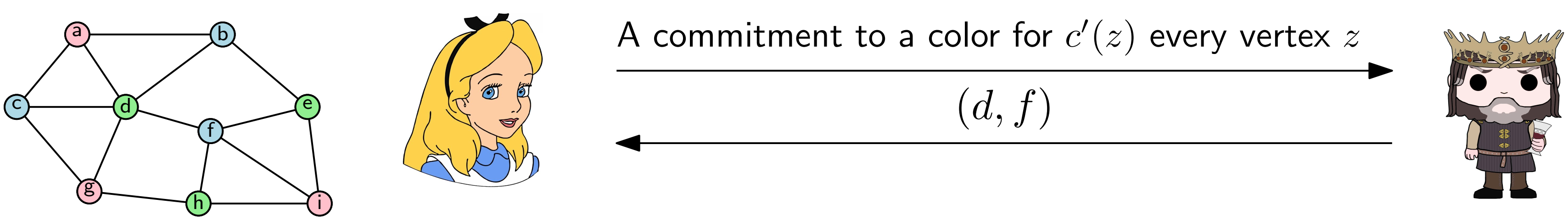
A CZK Proof System for Graph 3-Coloring

Common input: $G = (V, E)$

The prover knows a 3-coloring c of G

- The prover picks a random permutation π of $\{1, 2, 3\}$ and constructs the 3-coloring $c' = \pi \circ c$
- For each vertex v, the prover uses a commitment scheme to commit to $c'(v)$

  (commit to two bits for each vertex according to some encoding, e.g., 01 $\rightarrow 1$, 10 $\rightarrow 2$, $\{$00, 11$\} \rightarrow 3$)
- The verifier picks an edge (u^*, v^*) u.a.r. in E and sends (u^*, v^*) to the prover



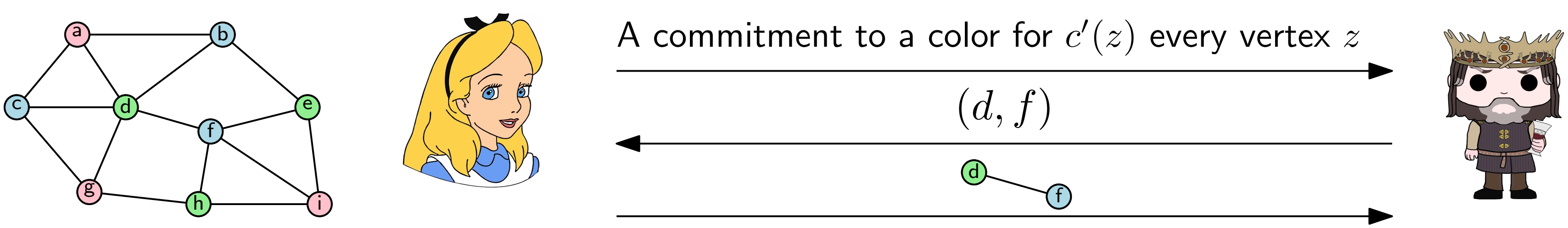
A CZK Proof System for Graph 3-Coloring

Common input: $G = (V, E)$

The prover knows a 3-coloring c of G

- The prover picks a random permutation π of $\{1, 2, 3\}$ and constructs the 3-coloring $c' = \pi \circ c$
- For each vertex v, the prover uses a commitment scheme to commit to $c'(v)$

  (commit to two bits for each vertex according to some encoding, e.g., 01 $\rightarrow 1$, 10 $\rightarrow 2$, $\{$00, 11$\} \rightarrow 3$)
- The verifier picks an edge (u^*, v^*) u.a.r. in E and sends (u^*, v^*) to the prover
- The prover opens the commitments for u and v and reveals $c'(u^*)$ and $c'(v^*)$



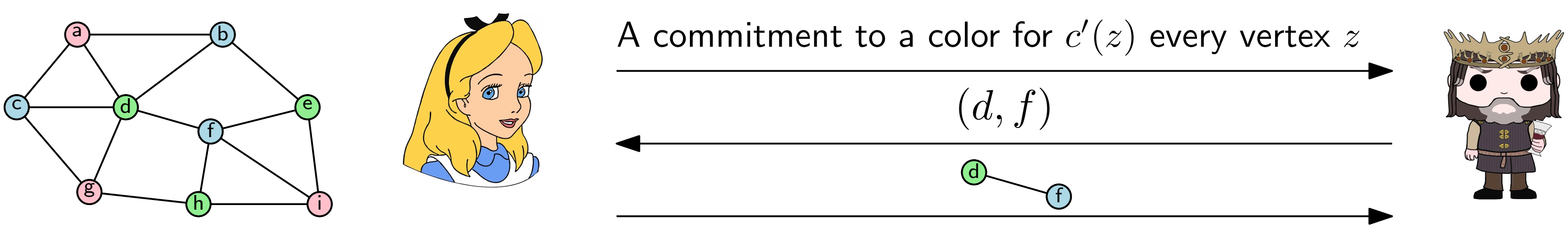
A CZK Proof System for Graph 3-Coloring

Common input: $G = (V, E)$

The prover knows a 3-coloring c of G

- The prover picks a random permutation π of $\{1, 2, 3\}$ and constructs the 3-coloring $c' = \pi \circ c$
- For each vertex v, the prover uses a commitment scheme to commit to $c'(v)$

  (commit to two bits for each vertex according to some encoding, e.g., 01 $\to 1$, 10 $\to 2$, $\{$00, 11$\} \to 3$)
- The verifier picks an edge (u^*, v^*) u.a.r. in E and sends (u^*, v^*) to the prover
- The prover opens the commitments for u and v and reveals $c'(u^*)$ and $c'(v^*)$
- The verifier accepts iff $c'(u^*) \neq c'(v^*)$



A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

Soundness:

- Consider a (possibly cheating) prover B and let c'' be the coloring resulting from its commitments
- Since G is not 3-colorable, there is at least one edge $(u, v) \in E$ such that $c''(u) = c''(v)$
- With probability at least $\frac{1}{|E|}$, the verifier picks $(u^*, v^*) = (u, v)$

A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

Soundness:

- Consider a (possibly cheating) prover B and let c'' be the coloring resulting from its commitments
- Since G is not 3-colorable, there is at least one edge $(u, v) \in E$ such that $c''(u) = c''(v)$
- With probability at least $\frac{1}{|E|}$, the verifier picks $(u^*, v^*) = (u, v)$
- When this happens, the only valid way to open the commitments for u^* and v^* is that of revealing $c''(u^*)$ and $c''(v^*)$
  (except for the negligible probability that the commitment of u^* or v^* was ambiguous)

A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

Soundness:

- Consider a (possibly cheating) prover B and let c'' be the coloring resulting from its commitments
- Since G is not 3-colorable, there is at least one edge $(u, v) \in E$ such that $c''(u) = c''(v)$
- With probability at least $\frac{1}{|E|}$, the verifier picks $(u^*, v^*) = (u, v)$
- When this happens, the only valid way to open the commitments for u^* and v^* is that of revealing $c''(u^*)$ and $c''(v^*)$
  (except for the negligible probability that the commitment of u^* or v^* was ambiguous)

$$\Pr[\text{the verifier accepts}] \leq 1 - \frac{1}{|E|} + \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

Soundness:

- Consider a (possibly cheating) prover B and let c'' be the coloring resulting from its commitments
- Since G is not 3-colorable, there is at least one edge $(u, v) \in E$ such that $c''(u) = c''(v)$
- With probability at least $\frac{1}{|E|}$, the verifier picks $(u^*, v^*) = (u, v)$
- When this happens, the only valid way to open the commitments for u^* and v^* is that of revealing $c''(u^*)$ and $c''(v^*)$
  (except for the negligible probability that the commitment of u^* or v^* was ambiguous)

$$\Pr[\text{the verifier accepts}] \leq 1 - \frac{1}{|E|} + \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

The gap between the probabilities is $\frac{1}{|E|} - \varepsilon(n) \geq \frac{1}{n^2}$ (for sufficiently large n)

A CZK Proof System for Graph 3-Coloring

Completeness:

- When c is a 3 coloring of G, $c(u^*) \neq c(v^*) \implies c'(u^*) \neq c'(v^*)$ and the verifier accepts

$$\Pr[\text{the verifier accepts}] = 1$$

Soundness:

- Consider a (possibly cheating) prover B and let c'' be the coloring resulting from its commitments
- Since G is not 3-colorable, there is at least one edge $(u, v) \in E$ such that $c''(u) = c''(v)$
- With probability at least $\frac{1}{|E|}$, the verifier picks $(u^*, v^*) = (u, v)$
- When this happens, the only valid way to open the commitments for u^* and v^* is that of revealing $c''(u^*)$ and $c''(v^*)$
  (except for the negligible probability that the commitment of u^* or v^* was ambiguous)

$$\Pr[\text{the verifier accepts}] \leq 1 - \frac{1}{|E|} + \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

The gap between the probabilities is $\frac{1}{|E|} - \varepsilon(n) \geq \frac{1}{n^2}$ (for sufficiently large n)

Use probability amplification

A CZK Proof System for Graph 3-Coloring

Zero-Knowledge:

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M_V^P(x)$**:**

- Choose (u^*, v^*) u.a.r. in E
- Choose two random distinct colors $c'(u^*), c'(v^*) \in \{1, 2, 3\}$ and a color $c'(z) \in \{1, 2, 3\}$ for each $z \in V \setminus \{u^*, v^*\}$ independently and u.a.r.

Note: c' may not be a valid coloring.

A CZK Proof System for Graph 3-Coloring

Zero-Knowledge:

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M_V^P(x)$**:**

- Choose (u^*, v^*) u.a.r. in E
- Choose two random distinct colors $c'(u^*), c'(v^*) \in \{1, 2, 3\}$ and a color $c'(z) \in \{1, 2, 3\}$ for each $z \in V \setminus \{u^*, v^*\}$ independently and u.a.r.
- Simulate the commitments of the prover to $c'(z)$ for all $z \in V$
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$

Note: c' may not be a valid coloring. We are using the hiding property of the commitment scheme

A CZK Proof System for Graph 3-Coloring

Zero-Knowledge:

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M_V^P(x)$**:**

- Choose (u^*, v^*) u.a.r. in E
- Choose two random distinct colors $c'(u^*), c'(v^*) \in \{1,2,3\}$ and a color $c'(z) \in \{1,2,3\}$ for each $z \in V \setminus \{u^*, v^*\}$ independently and u.a.r.
- Simulate the commitments of the prover to $c'(z)$ for all $z \in V$
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$
- The view consists of:

— All the simulated messages sent from the prover as part of the commitment and reveal phases of the commitment scheme

— All the random bits used by the verifier in the commitment phase of the commitment scheme

— (the random bits used to choose) the edge (u^*, v^*)

Note: c' may not be a valid coloring. We are using the hiding property of the commitment scheme

A CZK Proof System for Graph 3-Coloring

Zero-Knowledge: **How do we handle a cheating verifier V^*?**

It is easy to come up with a simulator for the interaction between P and the honest verifier V

Simulator $M_V^P(x)$:

- Choose (u^*, v^*) u.a.r. in E
- Choose two random distinct colors $c'(u^*), c'(v^*) \in \{1, 2, 3\}$ and a color $c'(z) \in \{1, 2, 3\}$ for each $z \in V \setminus \{u^*, v^*\}$ independently and u.a.r.
- Simulate the commitments of the prover to $c'(z)$ for all $z \in V$
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$
- The view consists of:

— All the simulated messages sent from the prover as part of the commitment and reveal phases of the commitment scheme

— All the random bits used by the verifier in the commitment phase of the commitment scheme

— (the random bits used to choose) the edge (u^*, v^*)

Note: c' may not be a valid coloring. We are using the hiding property of the commitment scheme

A CZK Proof System for Graph 3-Coloring

Simulator $M^P_{V^*}(x)$:

- Choose $(\widetilde{u}, \widetilde{v})$ u.a.r. in E
- Choose two random distinct colors $c'(\widetilde{u}), c'(\widetilde{v}) \in \{1, 2, 3\}$ and a color $c'(z) \in \{1, 2, 3\}$ for each $z \in V \setminus \{\widetilde{u}, \widetilde{v}\}$ independently and u.a.r.

A CZK Proof System for Graph 3-Coloring

Simulator $M^P_{V^*}(x)$**:**

- Choose $(\widetilde{u}, \widetilde{v})$ u.a.r. in E
- Choose two random distinct colors $c'(\widetilde{u}), c'(\widetilde{v}) \in \{1, 2, 3\}$ and a color $c'(z) \in \{1, 2, 3\}$ for each $z \in V \setminus \{\widetilde{u}, \widetilde{v}\}$ independently and u.a.r.
- Simulate the verifier $V^*(x)$ and the commitments of the prover to $c'(z)$ for all $z \in V$
- "Receive" the edge (u^*, v^*) that the simulated verifier sends to the prover
- If $(u^*, v^*) \neq (\widetilde{u}, \widetilde{v})$: halt the simulator and **fail**

A CZK Proof System for Graph 3-Coloring

Simulator $M^P_{V^*}(x)$:

- Choose $(\widetilde{u}, \widetilde{v})$ u.a.r. in E
- Choose two random distinct colors $c'(\widetilde{u}), c'(\widetilde{v}) \in \{1,2,3\}$ and a color $c'(z) \in \{1,2,3\}$ for each $z \in V \setminus \{\widetilde{u}, \widetilde{v}\}$ independently and u.a.r.
- Simulate the verifier $V^*(x)$ and the commitments of the prover to $c'(z)$ for all $z \in V$
- "Receive" the edge (u^*, v^*) that the simulated verifier sends to the prover
- If $(u^*, v^*) \neq (\widetilde{u}, \widetilde{v})$: halt the simulator and **fail**
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$ to the simulated verifier
- The view consists of:

— All the simulated messages sent from the prover as part of the commitment and reveal phases of the commitment scheme

— All the random bits used by the simulated execution of $V^*(x)$

Probability of failure?

A CZK Proof System for Graph 3-Coloring

Simulator $M^P_{V^*}(x)$:

- Choose $(\widetilde{u}, \widetilde{v})$ u.a.r. in E
- Choose two random distinct colors $c'(\widetilde{u}), c'(\widetilde{v}) \in \{1,2,3\}$ and a color $c'(z) \in \{1,2,3\}$ for each $z \in V \setminus \{\widetilde{u}, \widetilde{v}\}$ independently and u.a.r.
- Simulate the verifier $V^*(x)$ and the commitments of the prover to $c'(z)$ for all $z \in V$
- "Receive" the edge (u^*, v^*) that the simulated verifier sends to the prover
- If $(u^*, v^*) \neq (\widetilde{u}, \widetilde{v})$: halt the simulator and **fail**
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$ to the simulated verifier
- The view consists of:

— All the simulated messages sent from the prover as part of the commitment and reveal phases of the commitment scheme

— All the random bits used by the simulated execution of $V^*(x)$

Probability of failure? At most $1 - \frac{1}{|E|}$

A CZK Proof System for Graph 3-Coloring

Simulator $M^P_{V^*}(x)$**:**

- Choose $(\widetilde{u}, \widetilde{v})$ u.a.r. in E
- Choose two random distinct colors $c'(\widetilde{u}), c'(\widetilde{v}) \in \{1,2,3\}$ and a color $c'(z) \in \{1,2,3\}$ for each $z \in V \setminus \{\widetilde{u}, \widetilde{v}\}$ independently and u.a.r.
- Simulate the verifier $V^*(x)$ and the commitments of the prover to $c'(z)$ for all $z \in V$
- "Receive" the edge (u^*, v^*) that the simulated verifier sends to the prover
- If $(u^*, v^*) \neq (\widetilde{u}, \widetilde{v})$: halt the simulator and **fail**
- Open the commitments of u^* and v^* to reveal $c'(u^*)$ and $c'(v^*)$ to the simulated verifier
- The view consists of:

— All the simulated messages sent from the prover as part of the commitment and reveal phases of the commitment scheme

— All the random bits used by the simulated execution of $V^*(x)$

Probability of failure? At most $1 - \frac{1}{|E|}$ Run the simulator until it succeeds. Expected polynomial time

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\mathsf{NP} \subseteq \mathsf{CZK}$$

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\mathsf{NP} \subseteq \mathsf{CZK}$$

What about languages outside NP**?**

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\mathsf{NP} \subseteq \mathsf{CZK}$$

What about languages outside NP**?**

An example language that is in IP but might not be in NP is graph non-isomorphism

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\text{NP} \subseteq \text{CZK}$$

What about languages outside NP**?**

An example language that is in IP but might not be in NP is graph non-isomorphism

Can we transform the previous interactive proof system into a zero-knowledge proof system?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\text{NP} \subseteq \text{CZK}$$

What about languages outside NP**?**

An example language that is in IP but might not be in NP is graph non-isomorphism

Can we transform the previous interactive proof system into a zero-knowledge proof system?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

This is not a zero-knowledge protocol.

Why?

Zero-Knowledge Proof Systems for Languages not in NP?

We have shown that, **if PRGs exists**, then any language in NP admits a computational zero-knowledge proof system

$$\mathsf{NP} \subseteq \mathsf{CZK}$$

What about languages outside NP**?**

An example language that is in IP but might not be in NP is graph non-isomorphism

Can we transform the previous interactive proof system into a zero-knowledge proof system?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

This is not a zero-knowledge protocol. Why?

A cheating verifier interacting with an honest prover can learn whether an arbitrary graph G' is isomorphic to G_1

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

It is easy to come up with a simulator for a verifier that always picks a graph G' that is isomorphic to either G_1 or G_2, **and knows the correct answer**

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

It is easy to come up with a simulator for a verifier that always picks a graph G' that is isomorphic to either G_1 or G_2, **and knows the correct answer**

What about an arbitrary (cheating) verifier?

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \dots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

It is easy to come up with a simulator for a verifier that always picks a graph G' that is isomorphic to either G_1 or G_2, **and knows the correct answer**

What about an arbitrary (cheating) verifier?

Idea: If only the verifier could **convince the prover** that he knows to which graph G' is isomorphic to (without leaking the answer)...

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \dots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

It is easy to come up with a simulator for a verifier that always picks a graph G' that is isomorphic to either G_1 or G_2, **and knows the correct answer**

What about an arbitrary (cheating) verifier?

Idea: If only the verifier could **convince the prover** that he knows to which graph G' is isomorphic to (without leaking the answer)...

This can be done with a perfect zero-knowledge proof-of-knowledge!

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The verifier and the prover engage in a perfect zero-knowledge proof-of-knowledge interactive protocol (with reversed roles).
  The verifier convices the prover that he knows a b such that G' is isomorphic to G_b
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \ldots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The verifier and the prover engage in a perfect zero-knowledge proof-of-knowledge interactive protocol (with reversed roles).
  The verifier convices the prover that he knows a b such that G' is isomorphic to G_b
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

This implies that CZK might be larger than NP How big is CZK?

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \dots, n\}$

- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The verifier and the prover engage in a perfect zero-knowledge proof-of-knowledge interactive protocol (with reversed roles).
  The verifier convices the prover that he knows a b such that G' is isomorphic to G_b
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

This implies that CZK might be larger than NP   How big is CZK?

If PRGs exist:

- CZK = PSPACE

Zero-Knowledge Proof Systems for Languages not in NP?

Common input: $x = (G_1, G_2)$ where $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, and $V = \{1, \dots, n\}$

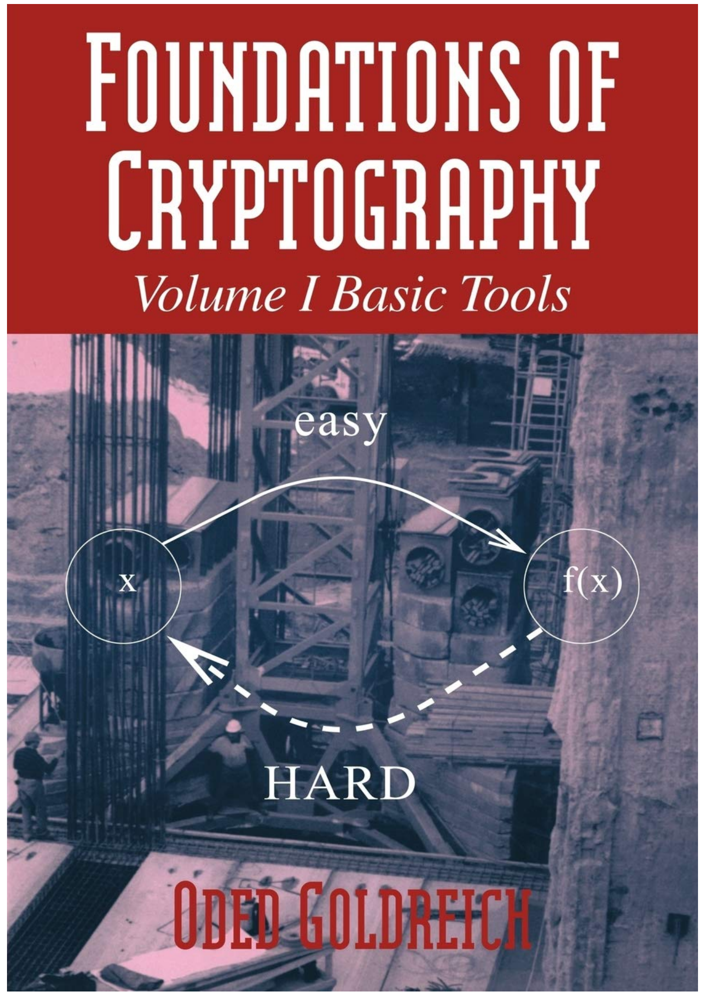
- The verifier chooses b u.a.r. in $\{1, 2\}$
- The verifier picks a random permutation $\pi : V \to V$ and sends $G' = \pi(G_b)$ to the prover
- The verifier and the prover engage in a perfect zero-knowledge proof-of-knowledge interactive protocol (with reversed roles).
  The verifier convices the prover that he knows a b such that G' is isomorphic to G_b
- The prover checks whether G' is isomorphic to G_1. If so it replies with $b' = 1$, otherwise it replies with $b' = 2$.
- If $b' = b$, the verifier accepts. Otherwise it rejects

This implies that CZK might be larger than NP   How big is CZK?

If PRGs exist:

- CZK = PSPACE
- It turns out that IP = PSPACE $\implies$ any language that admits an interactive proof system also admits a (computational) zero-knowledge proof system

References


FOUNDATIONS OF
CRYPTOGRAPHY
Volume I Basic Tools
easy
x
f(x)
HARD
ODED GOLDREICH