CS 594 – Advanced Crypto (Spring 2026)

Alex Block

Lecture 12

February 25, 2026

Types of Zero-knowledge

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs*.

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs*.
- Zero-knowledge:

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs.*

- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs.*

- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.
  - Honest-verifier vs. Dishonest-verifier.

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs.*

- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.
  - Honest-verifier vs. Dishonest-verifier.
  - Plain vs. Auxiliary-input.

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs*.
- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.
  - Honest-verifier vs. Dishonest-verifier.
  - Plain vs. Auxiliary-input.
- Statistical vs. Computational Soundness.

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs.*
- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.
  - Honest-verifier vs. Dishonest-verifier.
  - Plain vs. Auxiliary-input.
- Statistical vs. Computational Soundness.
- So when someone says they are using "zero-knowledge proofs," what do they mean?

Types of Zero-knowledge

- There are at roughly *24 notions of zero-knowledge proofs.*
- Zero-knowledge:
  - Perfect vs. Statistical vs. Computational.
  - Honest-verifier vs. Dishonest-verifier.
  - Plain vs. Auxiliary-input.
- Statistical vs. Computational Soundness.
- So when someone says they are using "zero-knowledge proofs," what do they mean?
- Usually, when people say "zero-knowledge proof," they mean the system at least has computational zero-knowledge and soundness, and has honest-verifier zero-knowledge.

Statistical Zero-knowledge Proofs

Limits of Statistical Zero-knowledge Proofs

Limits of Statistical Zero-knowledge Proofs

$\subseteq$ IP

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

- Requiring statistical zero-knowledge might seem desirable

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

- Requiring statistical zero-knowledge might seem desirable
  - Strong notion of privacy for the prover: privacy against unbounded algorithms!

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

- Requiring statistical zero-knowledge might seem desirable
  - Strong notion of privacy for the prover: privacy against unbounded algorithms!

- However, **SZK** is not that powerful.

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.
- Requiring statistical zero-knowledge might seem desirable
  - Strong notion of privacy for the prover: privacy against unbounded algorithms!
- However, **SZK** is not that powerful.

Theorem 1

$$\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}.$$

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

- Requiring statistical zero-knowledge might seem desirable
  - Strong notion of privacy for the prover: privacy against unbounded algorithms!

- However, **SZK** is not that powerful.

Theorem 1

$$\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}.$$

- **AM** is the set of all languages with 2-*message interactive proofs*, and **coAM** is its co-class.

Limits of Statistical Zero-knowledge Proofs

- Let **SZK** be the complexity class of all languages for which there exists a statistical zero-knowledge interactive proof of membership.

- Requiring statistical zero-knowledge might seem desirable
  - Strong notion of privacy for the prover: privacy against unbounded algorithms!

- However, **SZK** is not that powerful.

Theorem 1

$$\mathbf{SZK} \subseteq \mathbf{AM} \cap \mathbf{coAM}.$$

- **AM** is the set of all languages with 2-*message interactive proofs*, and **coAM** is its co-class.
  - A single message is sent first by the verifier, followed by the prover.

$L : L \in \mathrm{AM}$

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.
  - If it does, it would show that **AM** = **coAM**, which we believe is not true.

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.
  - If it does, it would show that **AM** = **coAM**, which we believe is not true.

- Limits the usefulness of **SZK**: cannot prove 3SAT in **SZK**!

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.
  - If it does, it would show that **AM** = **coAM**, which we believe is not true.

- Limits the usefulness of **SZK**: cannot prove 3SAT in **SZK**!
  - So we cannot use **SZK** as a general-purpose tool for arbitrary verifiable computation.

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.
  - If it does, it would show that **AM** = **coAM**, which we believe is not true.

- Limits the usefulness of **SZK**: cannot prove 3SAT in **SZK**!
  - So we cannot use **SZK** as a general-purpose tool for arbitrary verifiable computation.

- In contrast, if one-way functions exist, then **NP** $\subseteq$ **CZK** (computational zero-knowledge), and in fact **CZK** = **IP**.

Limits of Statistical Zero-knowledge Proofs

- We also do not believe **SZK** contains any **NP**-complete language.
  - If it does, it would show that **AM** = **coAM**, which we believe is not true.
- Limits the usefulness of **SZK**: cannot prove 3SAT in **SZK**!
  - So we cannot use **SZK** as a general-purpose tool for arbitrary verifiable computation.
- In contrast, if one-way functions exist, then **NP** $\subseteq$ **CZK** (computational zero-knowledge), and in fact **CZK** = **IP**.
- We will still examine some **SZK** proofs to convey the power and some intuition about zero-knowledge.

Graph (Non-)Isomorphism

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.
- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that

$G_0 \cong G_1$

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.

- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that
  - $(i, j) \in E_0$ if and only if $(\pi(i), \pi(j)) \in E_1$.

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.

- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that
  - $(i, j) \in E_0$ if and only if $(\pi(i), \pi(j)) \in E_1$.
  - We denote this as $\pi(G_0) = G_1$.

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.
- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that
  - $(i, j) \in E_0$ if and only if $(\pi(i), \pi(j)) \in E_1$.
  - We denote this as $\pi(G_0) = G_1$.
- Otherwise, we say that G_0 and G_1 are *not* isomorphic.

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.
- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that
  - $(i, j) \in E_0$ if and only if $(\pi(i), \pi(j)) \in E_1$.
  - We denote this as $\pi(G_0) = G_1$.
- Otherwise, we say that G_0 and G_1 are *not* isomorphic.
- There is no known poly-time algorithm for deciding if two graphs are isomorphic!

Graph (Non-)Isomorphism

- Let $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ be two graphs on n vertices.
- We say that G_0 is isomorphic to G_1 if there exists a permutation $\pi : [n] \to [n]$ such that
  - $(i, j) \in E_0$ if and only if $(\pi(i), \pi(j)) \in E_1$.
  - We denote this as $\pi(G_0) = G_1$.
- Otherwise, we say that G_0 and G_1 are *not* isomorphic.
- There is no known poly-time algorithm for deciding if two graphs are isomorphic!
- We give an honest-verifier SZK protocol (actually PZK) for both Graph Isomorphism and Non-Isomorphism.

HVPZK Proof for Graph Isomorphism

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

HVPZK Proof for Graph Isomorphism

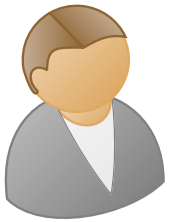
- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

(G_0, G_1)

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

(G_0, G_1)

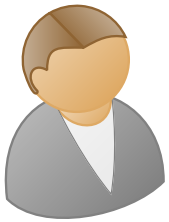


P

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

(G_0, G_1)

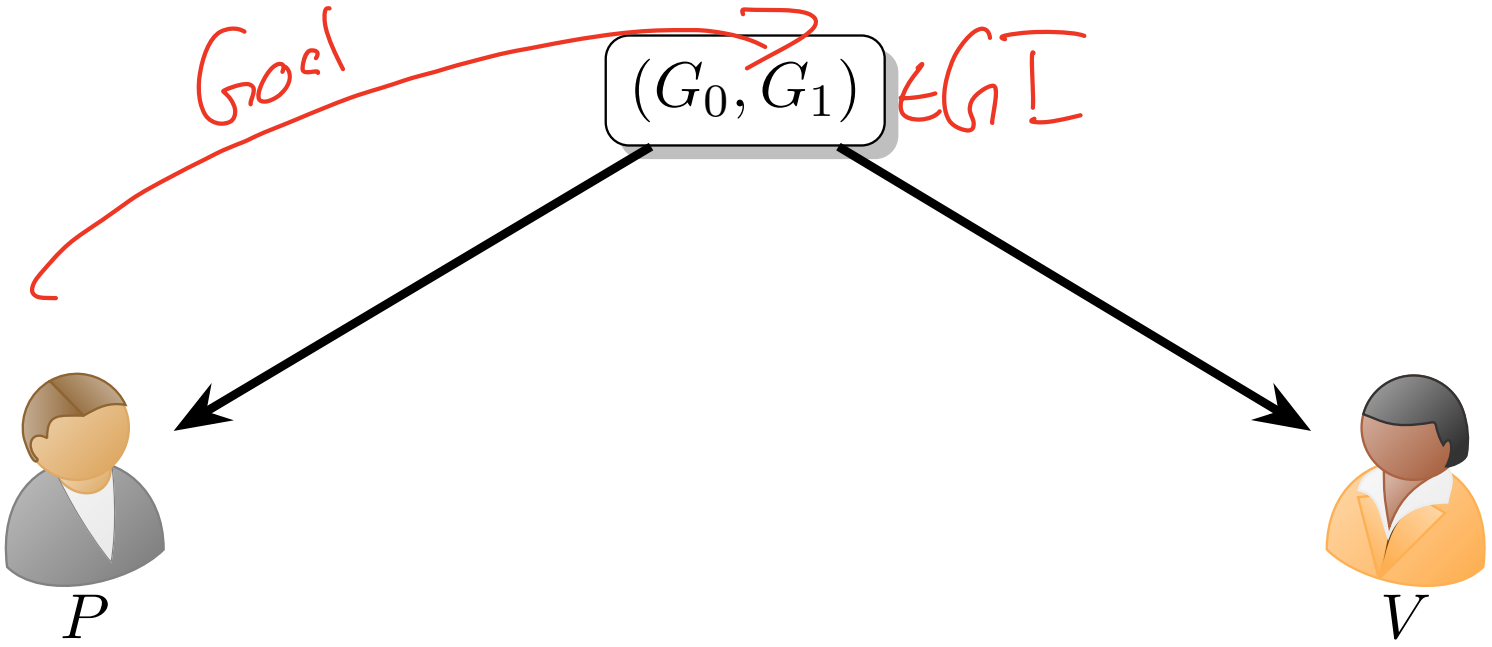


P

V

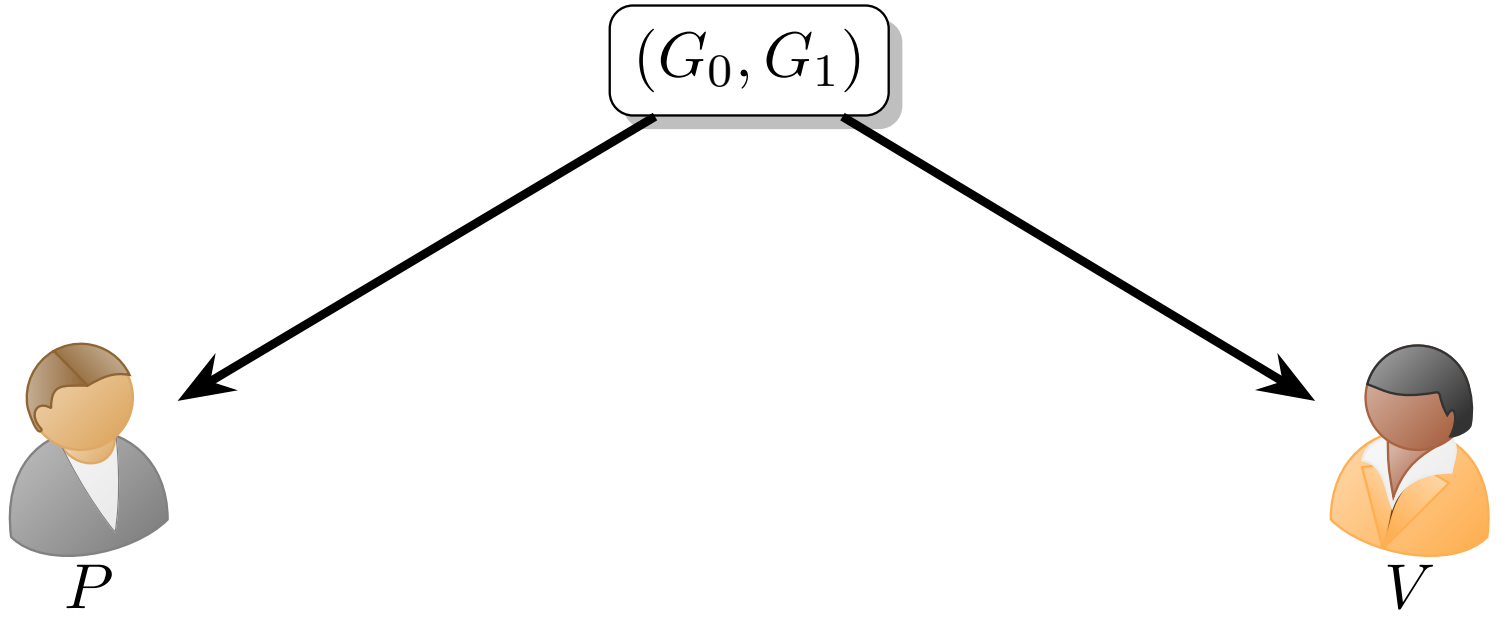
HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



HVPZK Proof for Graph Isomorphism

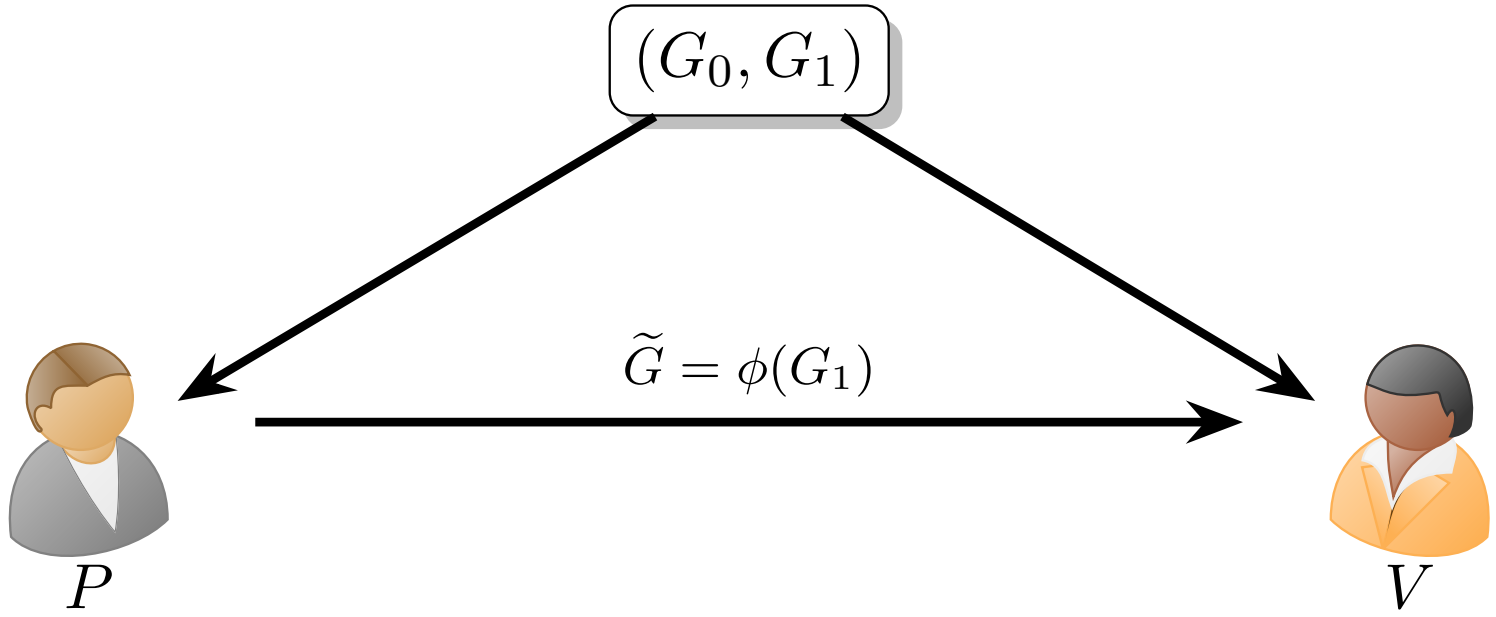
- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



(P1) Sample random permutation ϕ.

HVPZK Proof for Graph Isomorphism

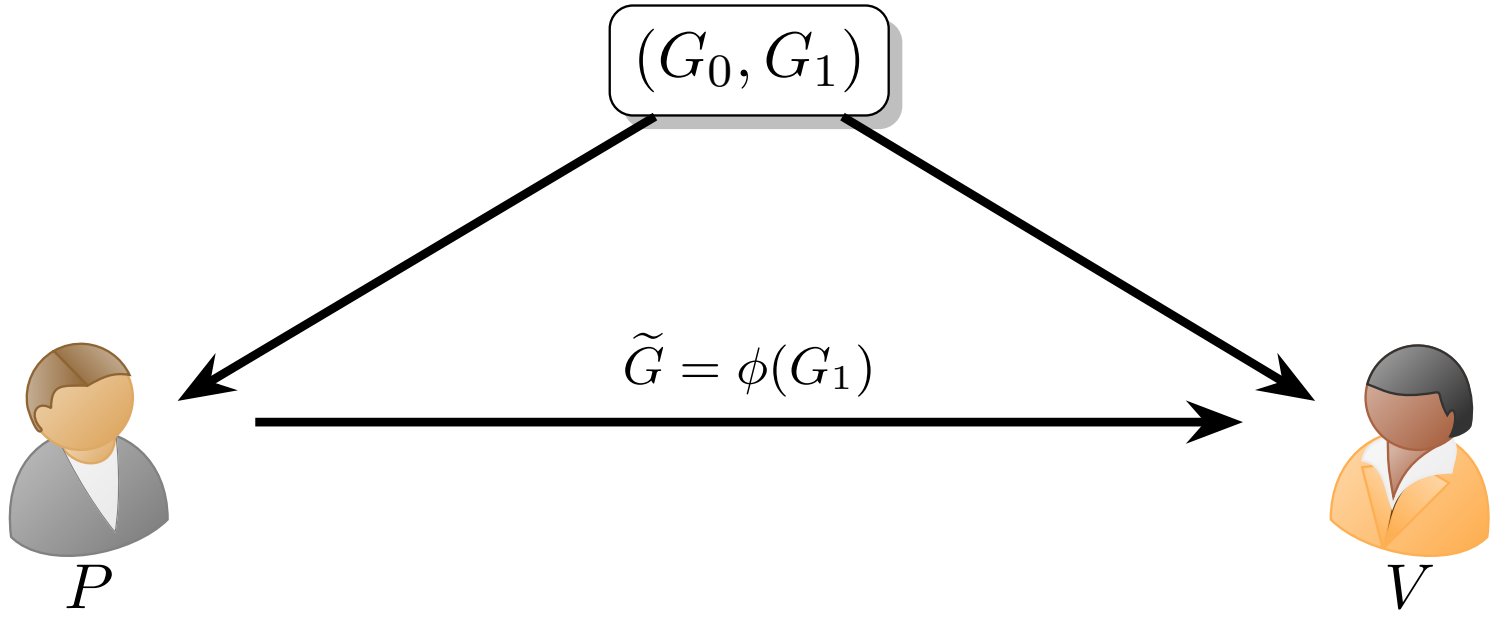
- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



(P1) Sample random permutation ϕ.

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

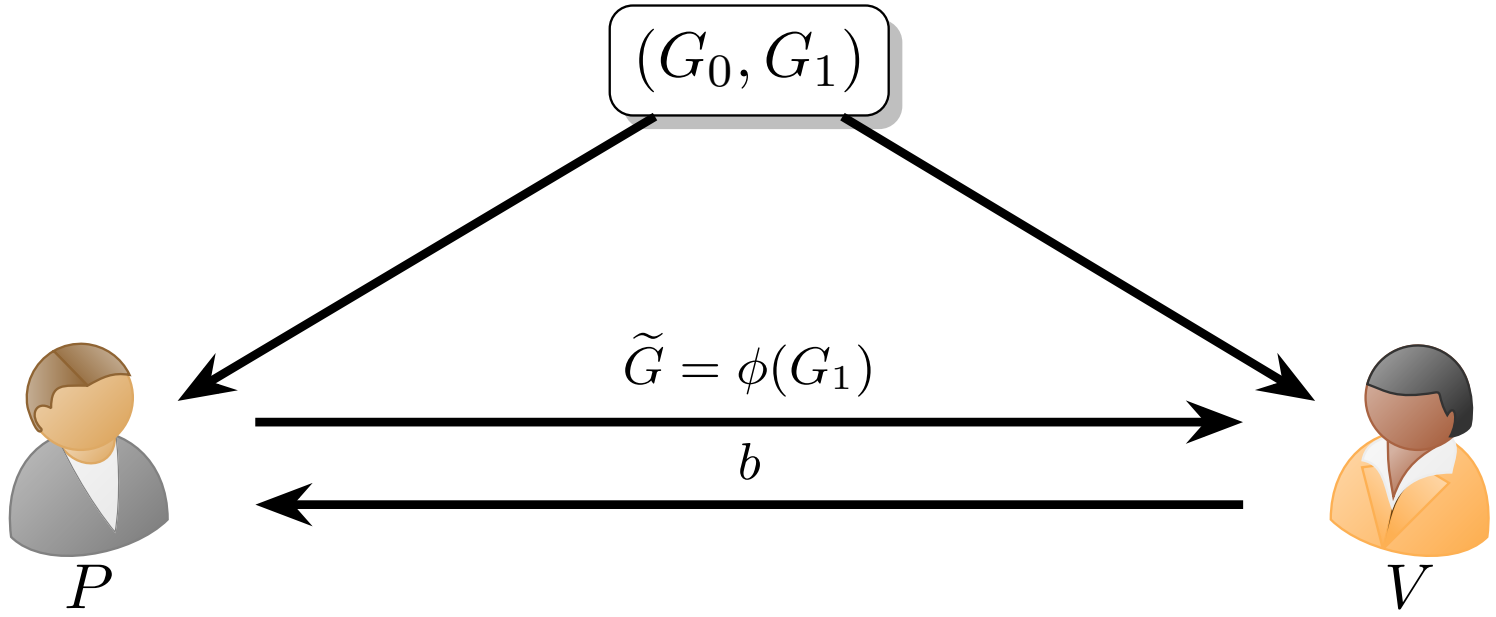


(P1) Sample random permutation ϕ.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$.

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.

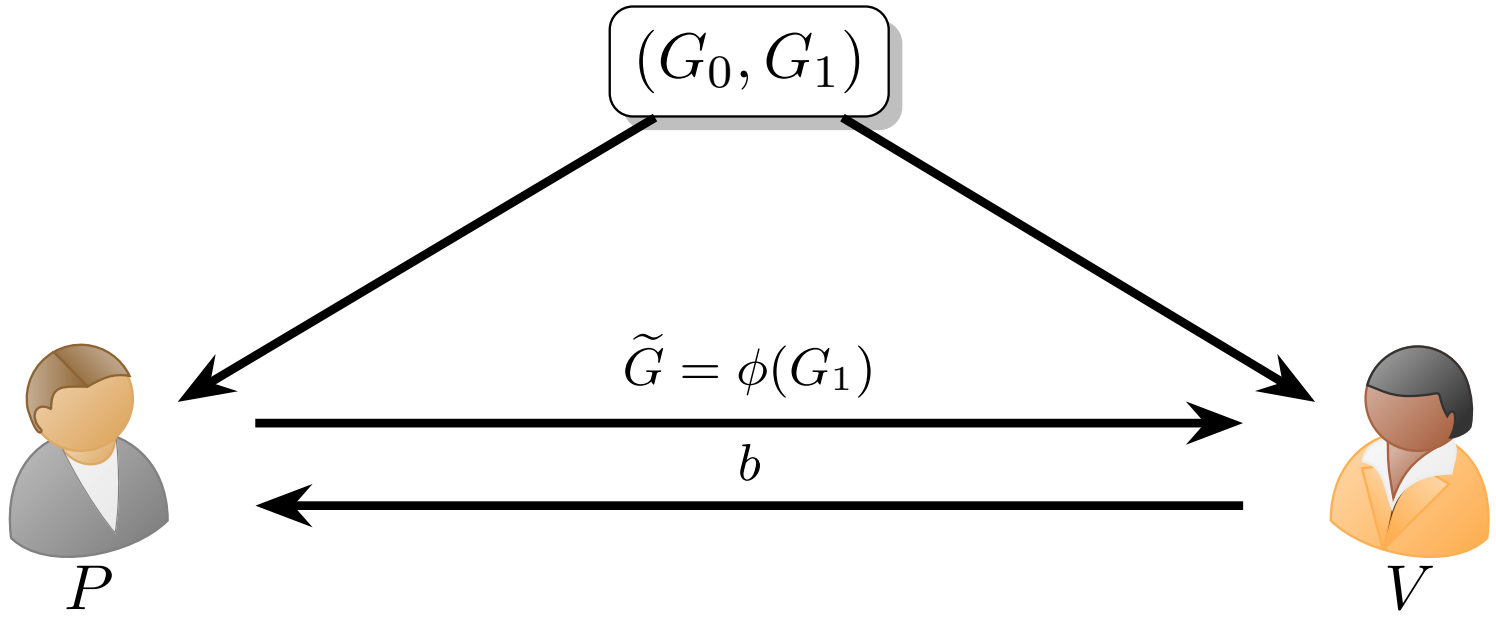


(P1) Sample random permutation ϕ.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$.

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



(P1) Sample random permutation ϕ.

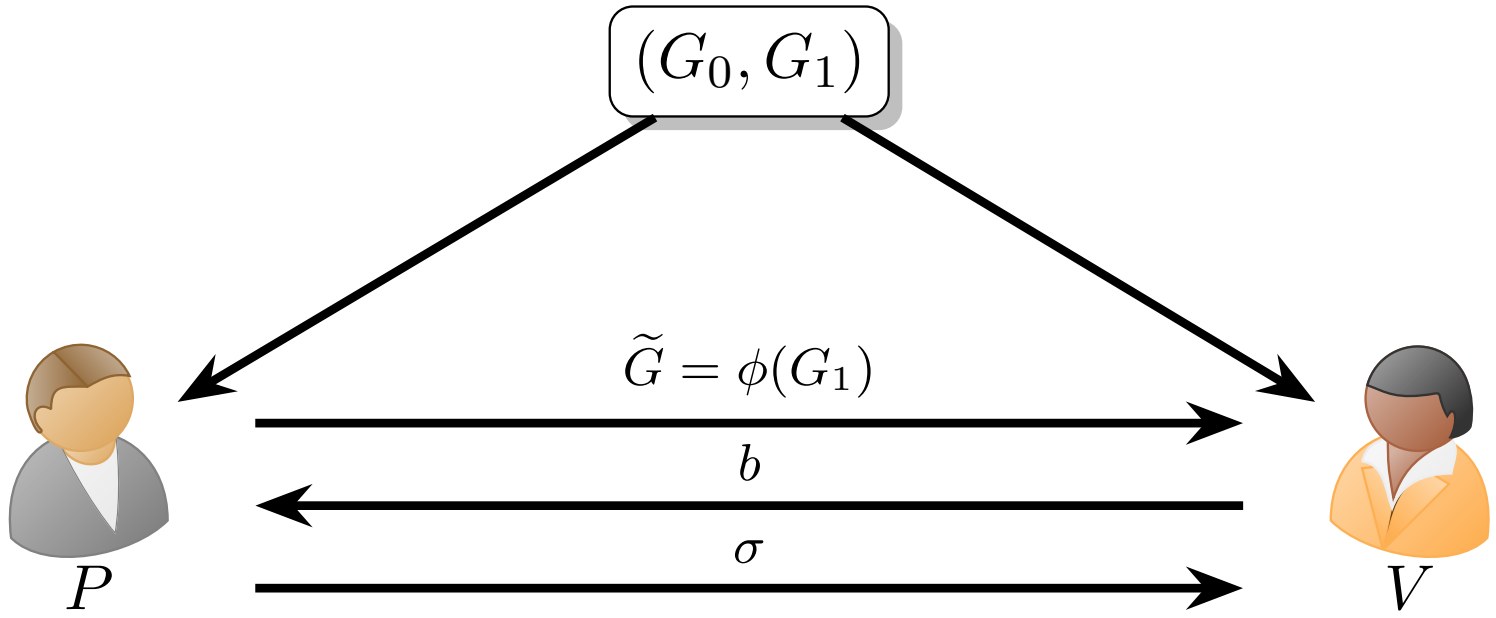
(V1) Sample $b \xleftarrow{\$} \{0, 1\}$.

(P2) Set $\sigma = \begin{cases} \phi & b = 1 \\ \phi \circ \pi & b = 0 \end{cases}$.

$(\phi \circ \pi)(x) = \phi(\pi(x))$

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



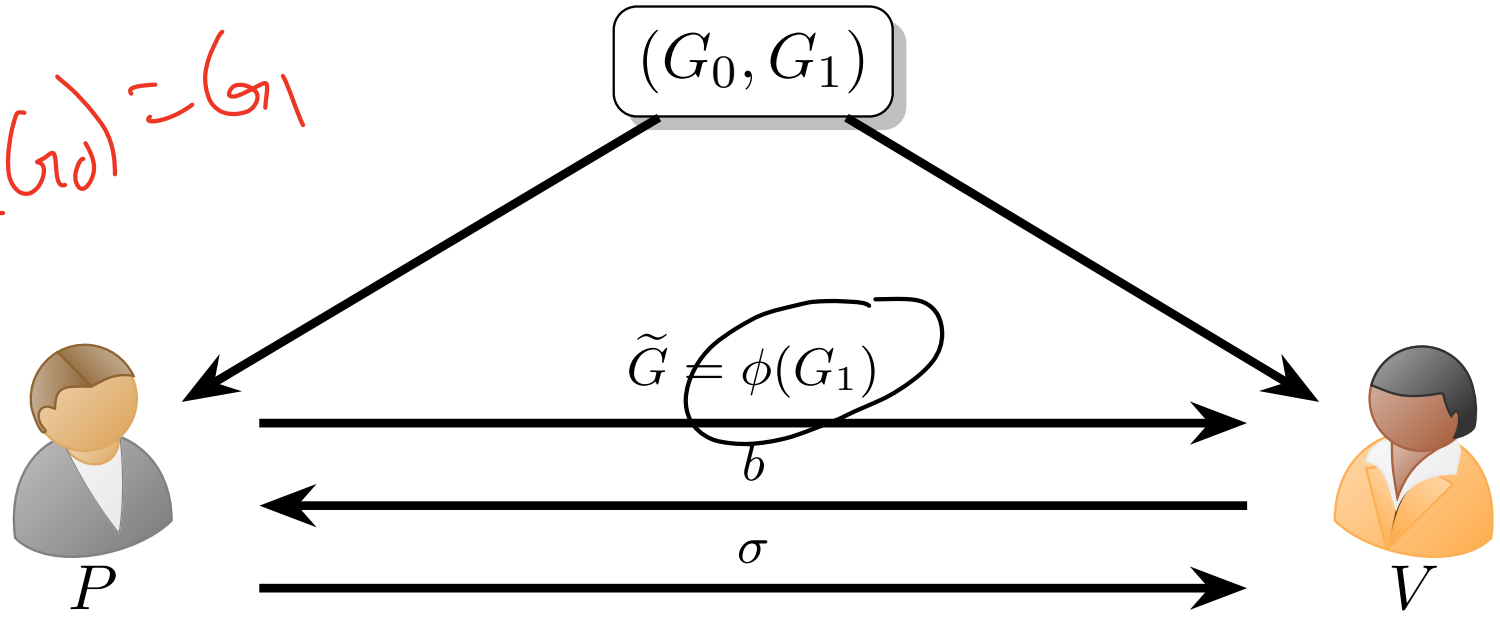
(P1) Sample random permutation ϕ.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$.

(P2) Set $\sigma = \begin{cases} \phi & b = 1 \\ \phi \circ \pi & b = 0 \end{cases}$.

HVPZK Proof for Graph Isomorphism

- Let $\mathbf{GI} = \{(G_0, G_1) \mid \exists \pi : \pi(G_0) = G_1\}$.



(P1) Sample random permutation ϕ.

(P2) Set $\sigma = \begin{cases} \phi & b = 1 \\ \phi \circ \pi & b = 0 \end{cases}$.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$.

(V2) Check if $\sigma(G_b) = \widetilde{G}$.

HZPZK Proof for Graph Isomorphism

HZPZK Proof for Graph Isomorphism

- Perfect Completeness

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.
  - Verifier will always accept.

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.
  - Verifier will always accept.

- Soundness error: 1/2.

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.
  - Verifier will always accept.

- Soundness error: $1/2$.
  - If G_0 and G_1 are not isomorphic, then prover will only convince verifier if $b = 1$.

$\tilde{G} = \phi(G_1)$

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.
  - Verifier will always accept.

- Soundness error: $1/2$.
  - If G_0 and G_1 are not isomorphic, then prover will only convince verifier if $b = 1$.
  - This happens with probability $1/2$.

HZPZK Proof for Graph Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are isomorphic via permutation π, then (computationally unbounded) prover can always answer the verifier correctly.
  - Verifier will always accept.

- Soundness error: $1/2$.
  - If G_0 and G_1 are not isomorphic, then prover will only convince verifier if $b = 1$.
  - This happens with probability $1/2$.
  - Sequentially compose this protocol twice to obtain soundness error $1/4 < 1/3$; 2^{-k} in general with k sequential compositions.

HZPZK Proof for Graph Isomorphism

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

$S(G_0, G_1)$:

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

$S(G_0, G_1)$:

- Sample $c \xleftarrow{\$} \{0, 1\}$.

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

$S(G_0, G_1)$:

- Sample $c \xleftarrow{\$} \{0, 1\}$.
- Sample random permutation φ.

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

$S(G_0, G_1)$:

- Sample $c \xleftarrow{\$} \{0, 1\}$.
- Sample random permutation φ.
- Set $H = \varphi(G_c)$.

HZPZK Proof for Graph Isomorphism

- Honest-verifier Perfect Zero-knowledge
  - Need to show that the *honest* verifier strategy can be perfectly simulated.

$S(G_0, G_1)$:

- Sample $c \xleftarrow{\$} \{0, 1\}$.
- Sample random permutation φ.
- Set $H = \varphi(G_c)$.
- Output (H, c, φ). $(\tilde{G}, b, \sigma)$

HZPZK Proof for Graph Isomorphism

Proof.

□

HZPZK Proof for Graph Isomorphism

Proof.

- Let $(H, c, \varphi) \leftarrow S(G_0, G_1)$.

HZPZK Proof for Graph Isomorphism

Proof.

- Let $(H, c, \varphi) \leftarrow S(G_0, G_1)$.
- If G_0 and G_1 are isomorphic, then this output is identically distributed as $(\widetilde{G}, b, \sigma)$ in the real execution of the protocol.

HZPZK Proof for Graph Isomorphism

Proof.

- Let $(H, c, \varphi) \leftarrow S(G_0, G_1)$.
- If G_0 and G_1 are isomorphic, then this output is identically distributed as $(\widetilde{G}, b, \sigma)$ in the real execution of the protocol.
- This is because $\widetilde{G}$ is isomorphic to both G_0 and G_1, and so is H, the bits b and c are uniformly sampled, and φ and σ both pass the verifier's check. $\square$

identically distributed

HZPZK Proof for Graph Isomorphism

Proof.

- Let $(H, c, \varphi) \leftarrow S(G_0, G_1)$.
- If G_0 and G_1 are isomorphic, then this output is identically distributed as $(\widetilde{G}, b, \sigma)$ in the real execution of the protocol.
- This is because $\widetilde{G}$ is isomorphic to both G_0 and G_1, and so is H, the bits b and c are uniformly sampled, and φ and σ both pass the verifier's check. □

- Can extend to *malicious* perfect zero-knowledge by allowing the simulator to run in *expected* polynomial time.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.
- If $b \neq c$, the simulator (being poly-time) cannot answer V^*, so it must try again.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.
- If $b \neq c$, the simulator (being poly-time) cannot answer V^*, so it must try again.
- The simulator succeeds with probability exactly $1/2$ whenever G_0 and G_1 are isomorphic.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.
- If $b \neq c$, the simulator (being poly-time) cannot answer V^*, so it must try again.
- The simulator succeeds with probability exactly $1/2$ whenever G_0 and G_1 are isomorphic.
  - This is because b cannot depend on c since φ is random.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.
- If $b \neq c$, the simulator (being poly-time) cannot answer V^*, so it must try again.
- The simulator succeeds with probability exactly $1/2$ whenever G_0 and G_1 are isomorphic.
  - This is because b cannot depend on c since φ is random.
- Expected number of loops before halting: 2.

PZK Proof for Graph Isomorphism

- Let V^* be an arbitrary PPT verifier strategy.

$S(G_0, G_1, V^*)$:

1. Sample $c \xleftarrow{\$} \{0,1\}$ and random permutation φ.
2. Set $H = \varphi(G_c)$ and $b \leftarrow V^*(G_0, G_1, H)$.
3. If $b = c$, output (H, b, φ), otherwise goto (1).

- Here, V^* can deviate arbitrarily from the protocol, but is still expected to send a bit b.
- If $b \neq c$, the simulator (being poly-time) cannot answer V^*, so it must try again.
- The simulator succeeds with probability exactly $1/2$ whenever G_0 and G_1 are isomorphic.
  - This is because b cannot depend on c since φ is random.
- Expected number of loops before halting: 2.
- Implies expected poly-time (running V^* is poly-time).

HVPZK Proof for Graph Non-Isomorphism

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) \colon \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi \colon \pi(G_0) \neq G_1\}$.

HVPZK Proof for Graph Non-Isomorphism

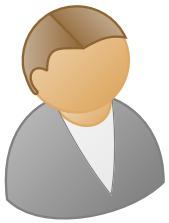
- Let $\mathbf{GNI} = \{(G_0, G_1) \colon \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi \colon \pi(G_0) \neq G_1\}$.

(G_0, G_1)

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) \colon \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi \colon \pi(G_0) \neq G_1\}$.

(G_0, G_1)

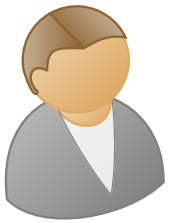


P

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi : \pi(G_0) \neq G_1\}$.

(G_0, G_1)



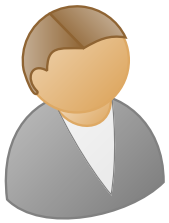
P

V

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi : \pi(G_0) \neq G_1\}$.

(G_0, G_1)

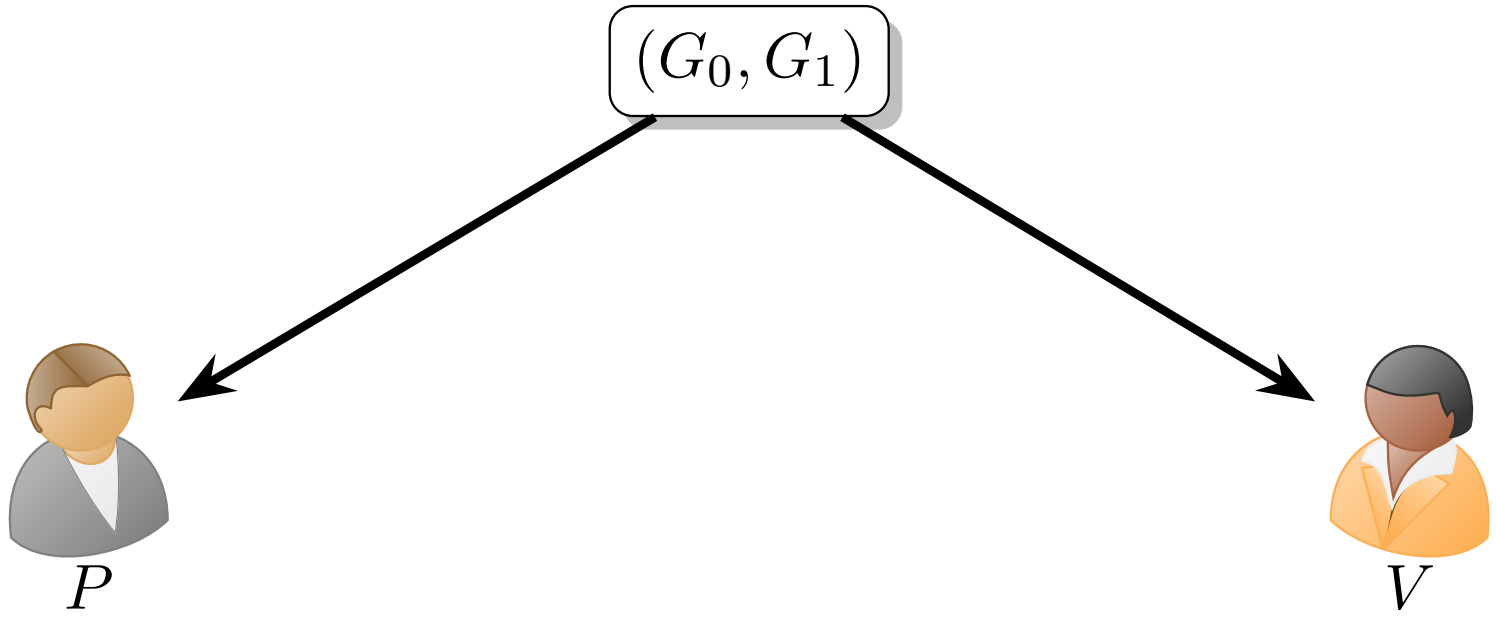


P

V

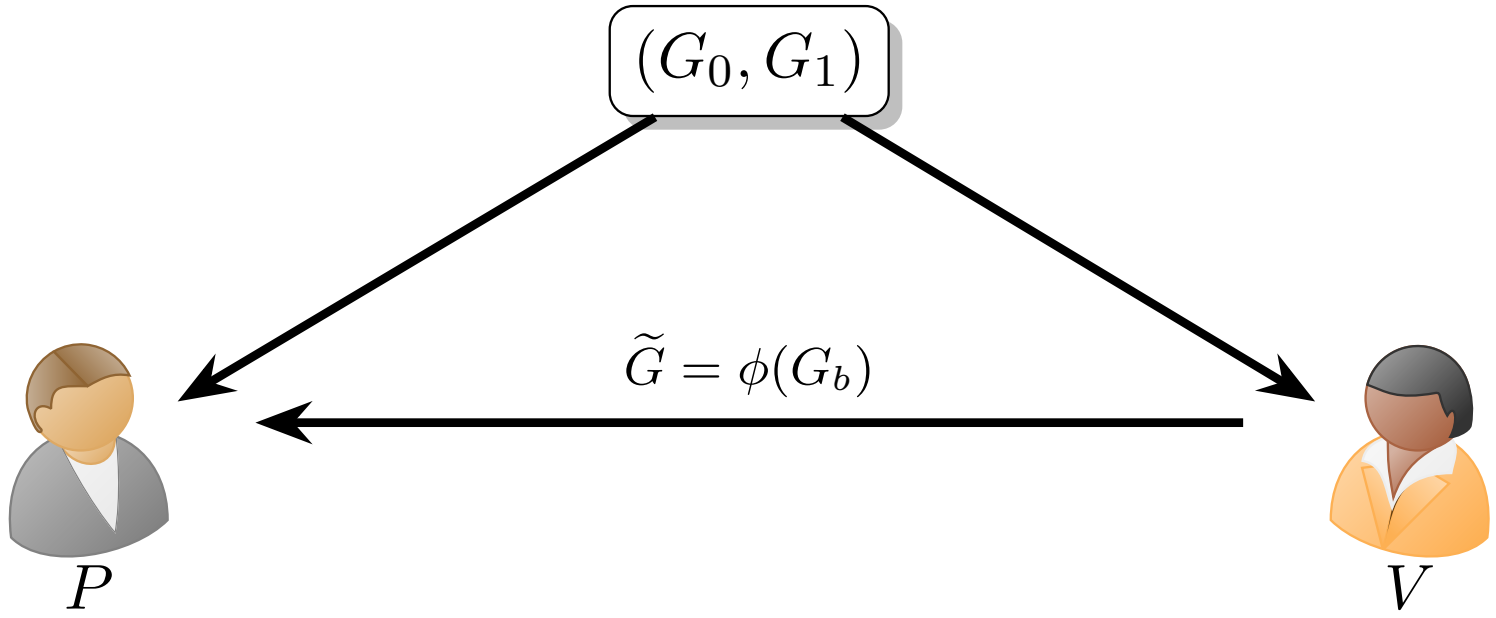
HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi : \pi(G_0) \neq G_1\}$.



HVPZK Proof for Graph Non-Isomorphism

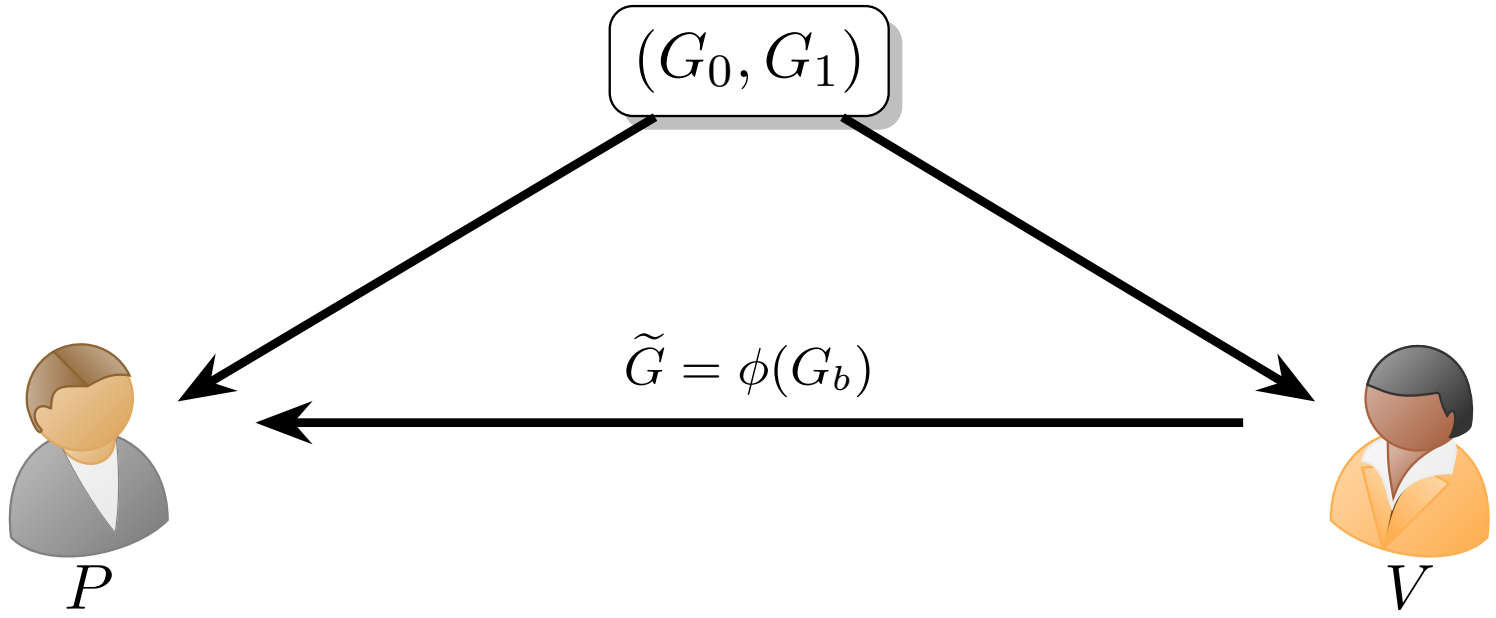
- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi : \pi(G_0) \neq G_1\}$.



(V1) Sample $b \xleftarrow{\$} \{0, 1\}$ and random permutation ϕ.

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) : \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi : \pi(G_0) \neq G_1\}$.

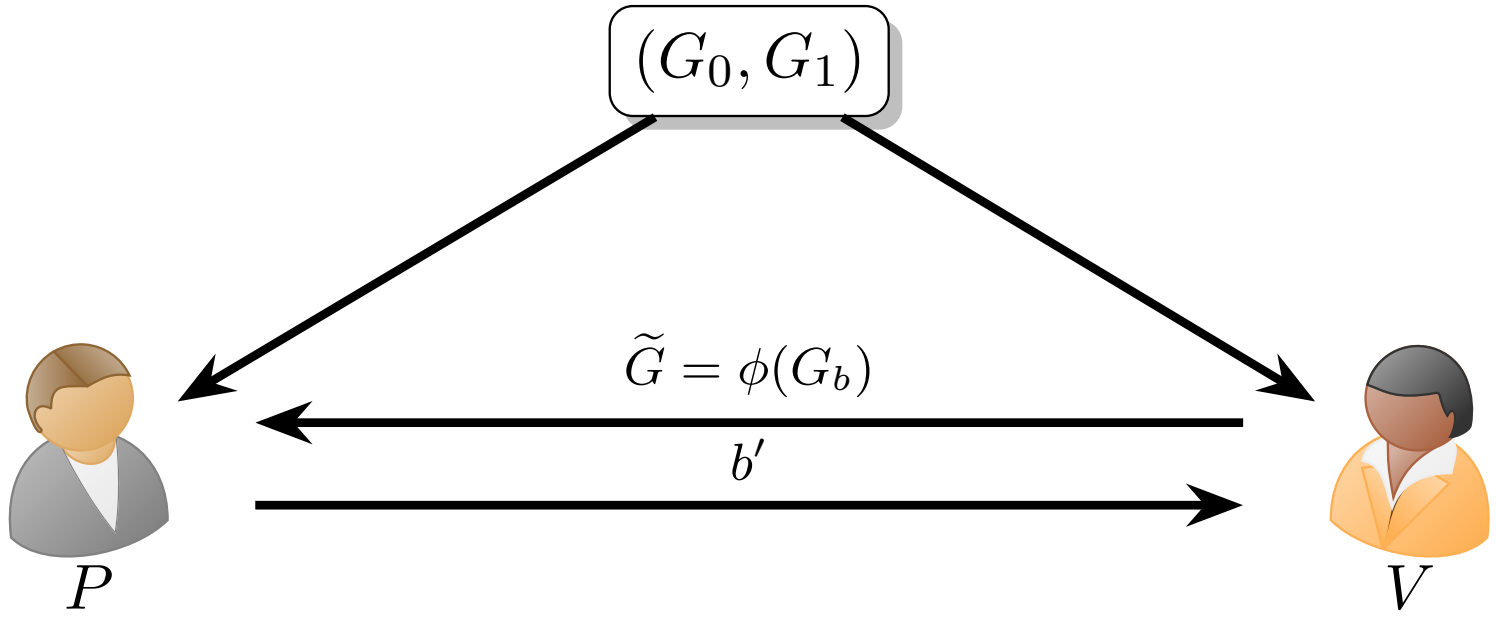


(P1) Find permutation σ and b' s.t. $\sigma(G_{b'}) = \widetilde{G}$.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$ and random permutation ϕ.

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) \colon \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi \colon \pi(G_0) \neq G_1\}$.

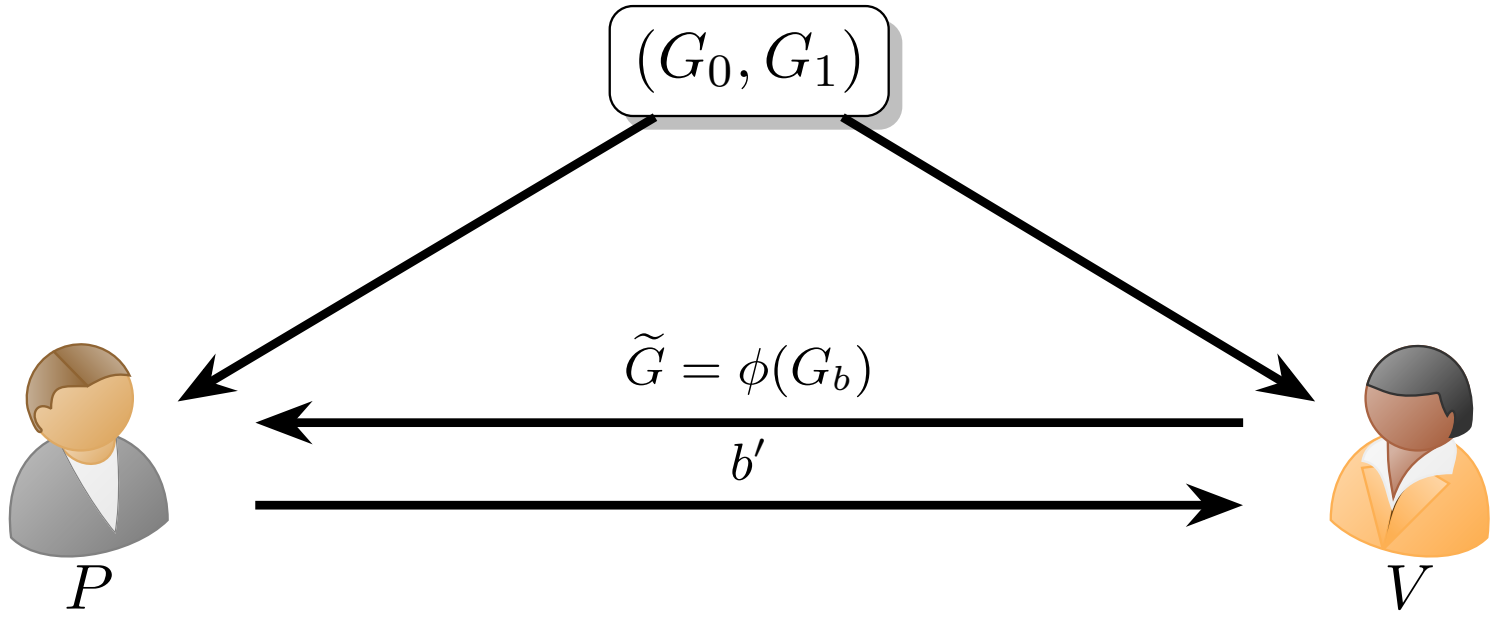


(P1) Find permutation σ and b' s.t. $\sigma(G_{b'}) = \widetilde{G}$.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$ and random permutation ϕ.

HVPZK Proof for Graph Non-Isomorphism

- Let $\mathbf{GNI} = \{(G_0, G_1) \colon \ G_0 \text{ and } G_1 \text{ are not isomorphic}\}$.
  - Equivalently: $\mathbf{GNI} = \{(G_0, G_1) \mid \forall \pi \colon \pi(G_0) \neq G_1\}$.



(P1) Find permutation σ and b' s.t. $\sigma(G_{b'}) = \widetilde{G}$.

(V1) Sample $b \xleftarrow{\$} \{0, 1\}$ and random permutation ϕ.

(V2) Check if $b = b'$.

HVPZK Proof for Graph Non-Isomorphism

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

- Soundness error: 1/2.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

- Soundness error: 1/2.
  - If G_0 and G_1 are isomorphic, then prover will not know which b was sampled to compute $\widetilde{G}$.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

- Soundness error: 1/2.
  - If G_0 and G_1 are isomorphic, then prover will not know which b was sampled to compute $\widetilde{G}$.
  - This is because G_0 and G_1 are both isomorphic to $\widetilde{G}$.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

- Soundness error: 1/2.
  - If G_0 and G_1 are isomorphic, then prover will not know which b was sampled to compute $\widetilde{G}$.
  - This is because G_0 and G_1 are both isomorphic to $\widetilde{G}$.
  - Prover cannot to better than guessing b', which succeeds with probability at most 1/2.

HVPZK Proof for Graph Non-Isomorphism

- Perfect Completeness
  - If G_0 and G_1 are not isomorphic, then the prover can always choose the correct bit $b' = b$ by finding which $G_{b'}$ is isomorphic to $\widetilde{G}$.
  - Only one of G_0 or G_1 will be isomorphic to $\widetilde{G}$.
  - Thus, the verifier will always accept.

- Soundness error: $1/2$.
  - If G_0 and G_1 are isomorphic, then prover will not know which b was sampled to compute $\widetilde{G}$.
  - This is because G_0 and G_1 are both isomorphic to $\widetilde{G}$.
  - Prover cannot to better than guessing b', which succeeds with probability at most $1/2$.
  - Sequentially compose this protocol twice to obtain soundness error $1/4 < 1/3$; 2^{-k} in general with k sequential compositions.

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

- Sample $b \xleftarrow{\$} \{0, 1\}$ and random permutation φ.

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

- Sample $b \xleftarrow{\$} \{0,1\}$ and random permutation φ.
- Output $(H = \varphi(G_b), b)$.

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

- Sample $b \xleftarrow{\$} \{0,1\}$ and random permutation φ.
- Output $(H = \varphi(G_b), b)$.

Proof.

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

- Sample $b \xleftarrow{\$} \{0,1\}$ and random permutation φ.
- Output $(H = \varphi(G_b), b)$.

Proof.

- Note that H and b are identically generated as $\widetilde{G}$ and b in the actual protocol.

HVPZK Proof for Graph Non-Isomorphism

- Honest-verifier Perfect Zero-knowledge

$S(G_0, G_1)$

- Sample $b \xleftarrow{\$} \{0,1\}$ and random permutation φ.
- Output $(H = \varphi(G_b), b)$.

Proof.

- Note that H and b are identically generated as $\widetilde{G}$ and b in the actual protocol.
- Moreover, if the graphs are not isomorphic, an honest prover always outputs $b' = b$. □

PZK Proof for Graph Non-Isomorphism?

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.
  - Suppose V^* has a graph H which is isomorphic to either G_0 or G_1, but it doesn't know which.

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.
  - Suppose V^* has a graph H which is isomorphic to either G_0 or G_1, but it doesn't know which.
  - V^* replaces $\widetilde{G}$ in the protocol with H.

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.
  - Suppose V^* has a graph H which is isomorphic to either G_0 or G_1, but it doesn't know which.
  - V^* replaces $\widetilde{G}$ in the protocol with H.
  - The honest prover responds with the correct bit b', and the verifier learns that H is isomorphic to $G_{b'}$.

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.
  - Suppose V^* has a graph H which is isomorphic to either G_0 or G_1, but it doesn't know which.
  - V^* replaces $\widetilde{G}$ in the protocol with H.
  - The honest prover responds with the correct bit b', and the verifier learns that H is isomorphic to $G_{b'}$.
  - This is something the verifier *cannot* learn on its own (since it is poly-time).

PZK Proof for Graph Non-Isomorphism?

- One might try to argue that the protocol satisfies *malicious verifier* zero-knowledge as well.
  - However, this is not the case!

- Consider the following strategy V^*.
  - Suppose V^* has a graph H which is isomorphic to either G_0 or G_1, but it doesn't know which.
  - V^* replaces $\widetilde{G}$ in the protocol with H.
  - The honest prover responds with the correct bit b', and the verifier learns that H is isomorphic to $G_{b'}$.
  - This is something the verifier *cannot* learn on its own (since it is poly-time).

- High-level fix: have the verifier *prove* it knows which graph $\widetilde{G}$ is isomorphic to, without revealing sampled bit b.

HVPZK Argument: Schnorr Identification Protocol

HVPZK Argument: Schnorr Identification Protocol

- Our final example of HVSZK will be a zero-knowledge *argument*, due to Schnorr.

HVPZK Argument: Schnorr Identification Protocol

- Our final example of HVSZK will be a zero-knowledge *argument*, due to Schnorr.

- Let $(\mathbb{G}, p, g)$ be the description of a prime-order cyclic group of size p with generator g.

HVPZK Argument: Schnorr Identification Protocol

- Our final example of HVSZK will be a zero-knowledge *argument*, due to Schnorr.
- Let $(\mathbb{G}, p, g)$ be the description of a prime-order cyclic group of size p with generator g.
- Schnorr's Identification Protocol allows a prover to convince a verifier it knows the discrete-log of some public element $A \in \mathbb{G}$.

HVPZK Argument: Schnorr Identification Protocol

- Our final example of HVSZK will be a zero-knowledge *argument*, due to Schnorr.
- Let $(\mathbb{G}, p, g)$ be the description of a prime-order cyclic group of size p with generator g.
- Schnorr's Identification Protocol allows a prover to convince a verifier it knows the discrete-log of some public element $A \in \mathbb{G}$.
- Formally, the protocol is an interactive argument for the language $L_{\mathsf{Sch}} = \{(\mathbb{G}, p, g, A) \mid \exists a \colon g^a = A\}$.

Schnorr Identification Protocol

Schnorr Identification Protocol

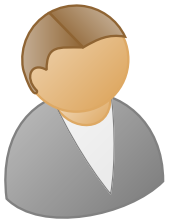
$(\mathbb{G}, p, g, A; a)$

Private

$\in R_{Sch}$

Schnorr Identification Protocol

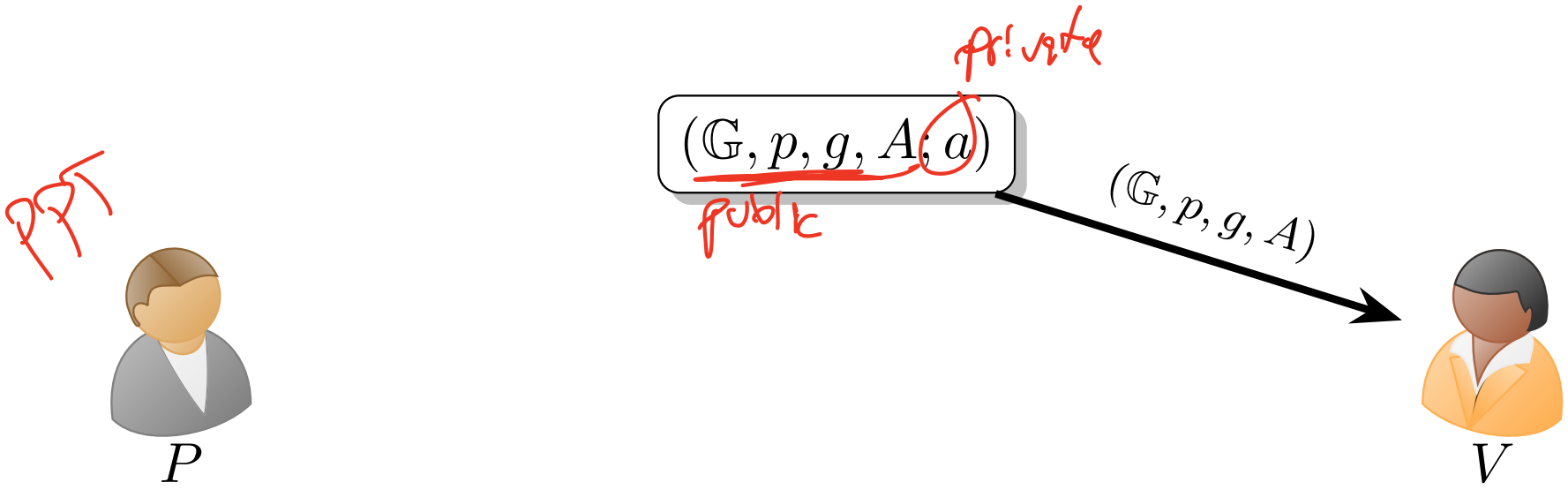
$(\mathbb{G}, p, g, A; a)$



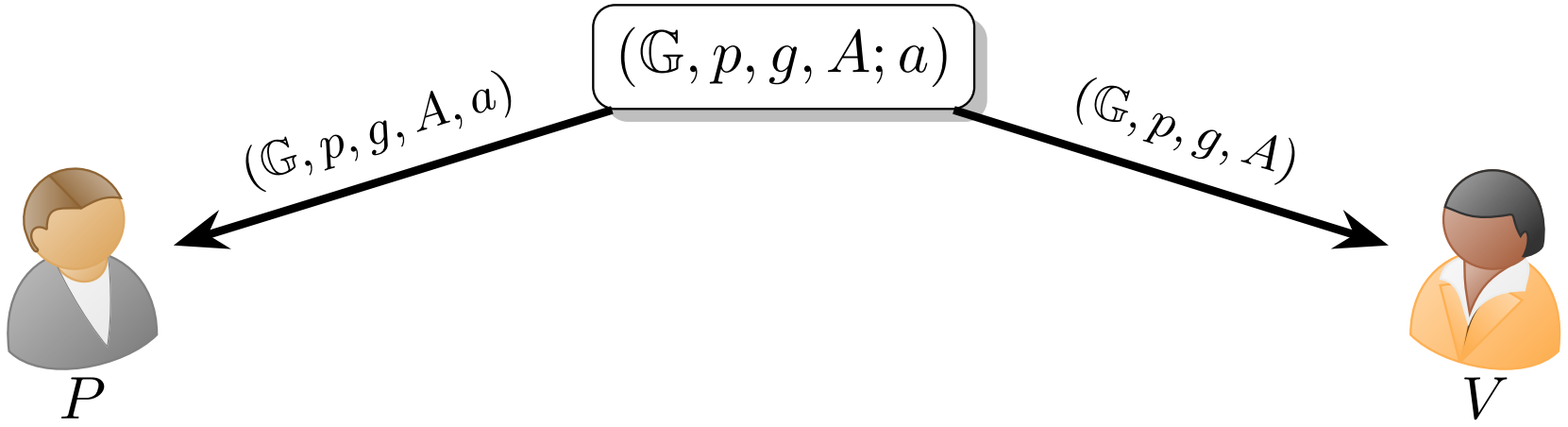
P

V

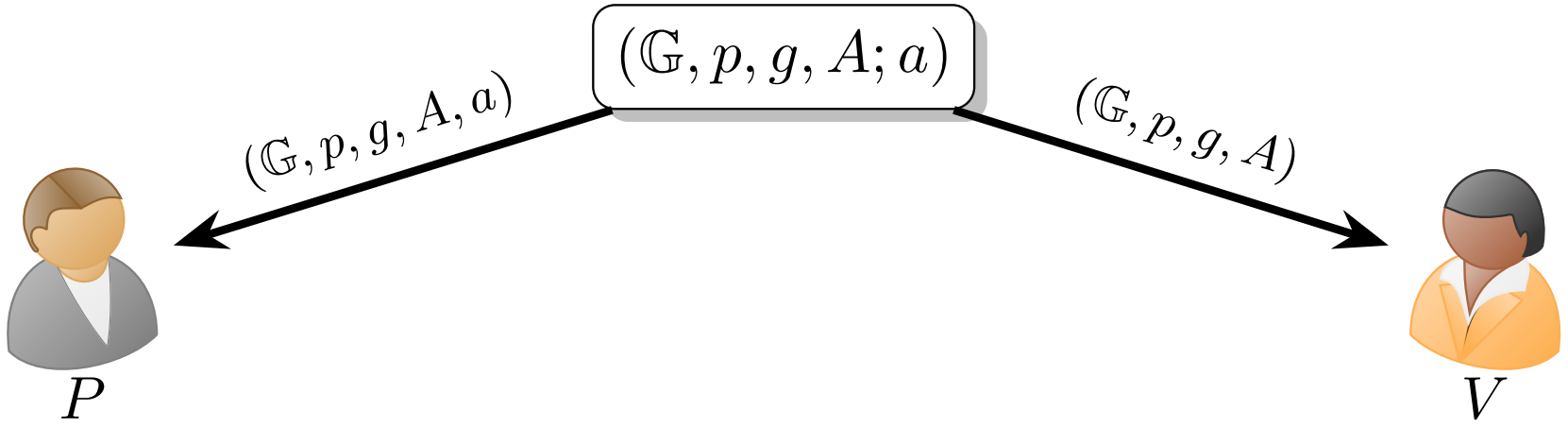
Schnorr Identification Protocol



Schnorr Identification Protocol

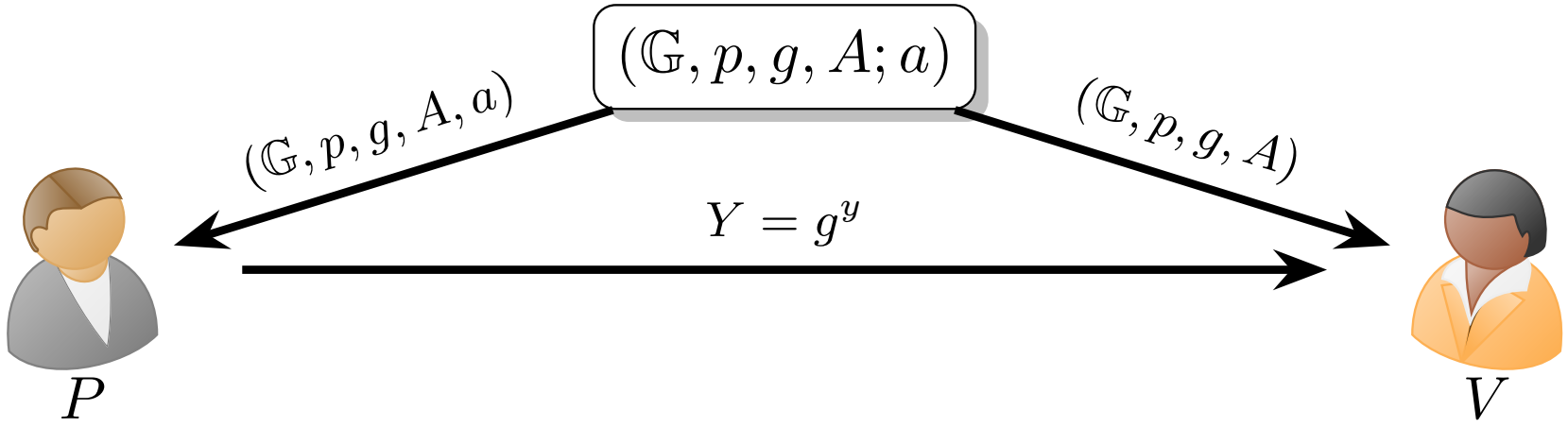


Schnorr Identification Protocol



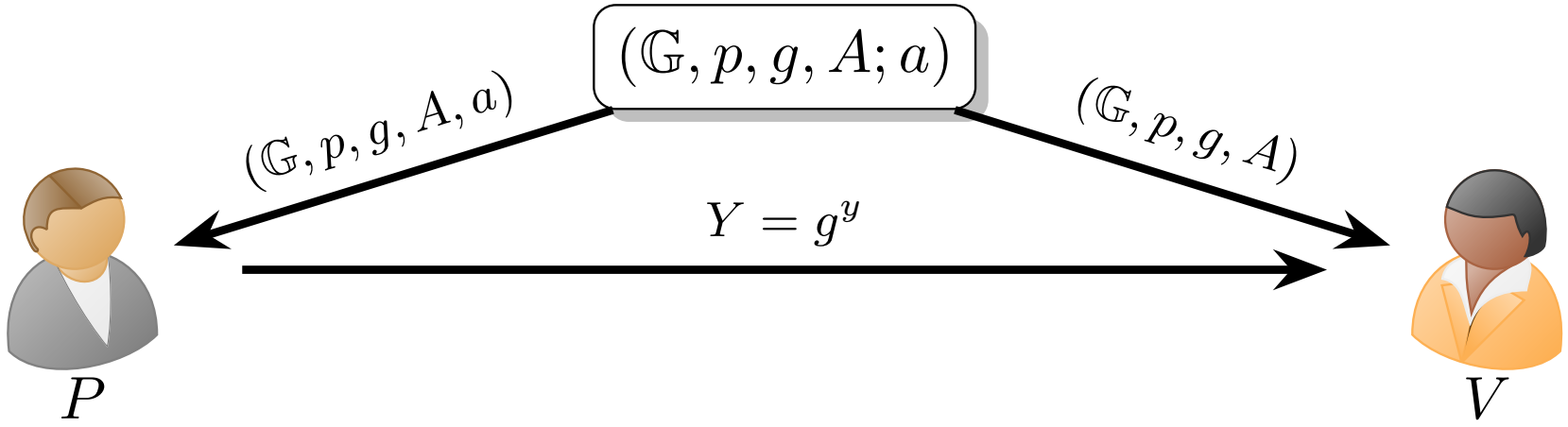
(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

Schnorr Identification Protocol



(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

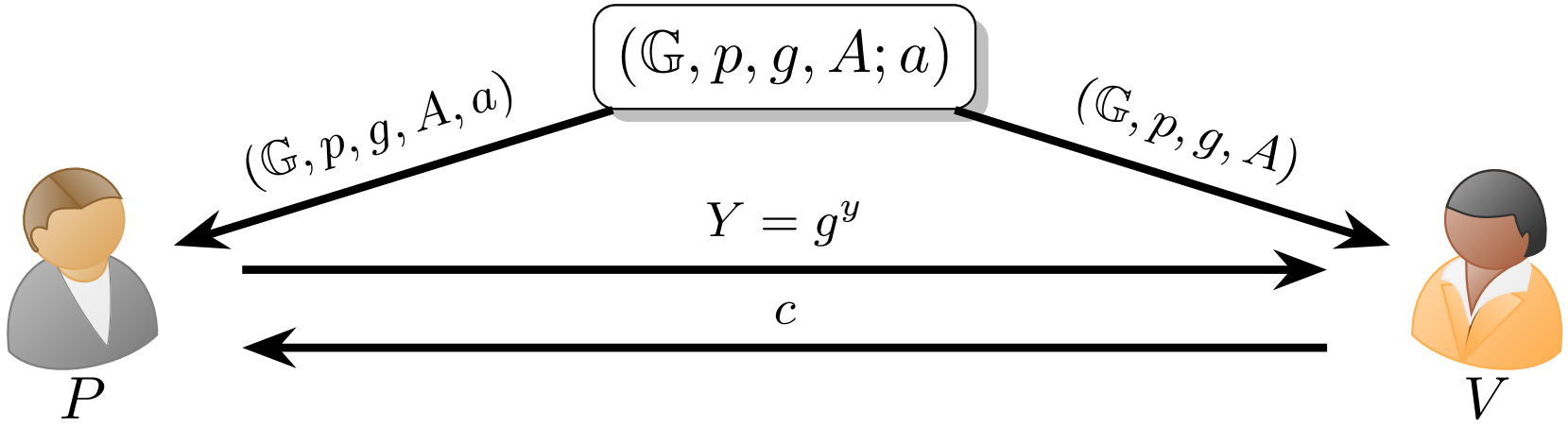
Schnorr Identification Protocol



(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

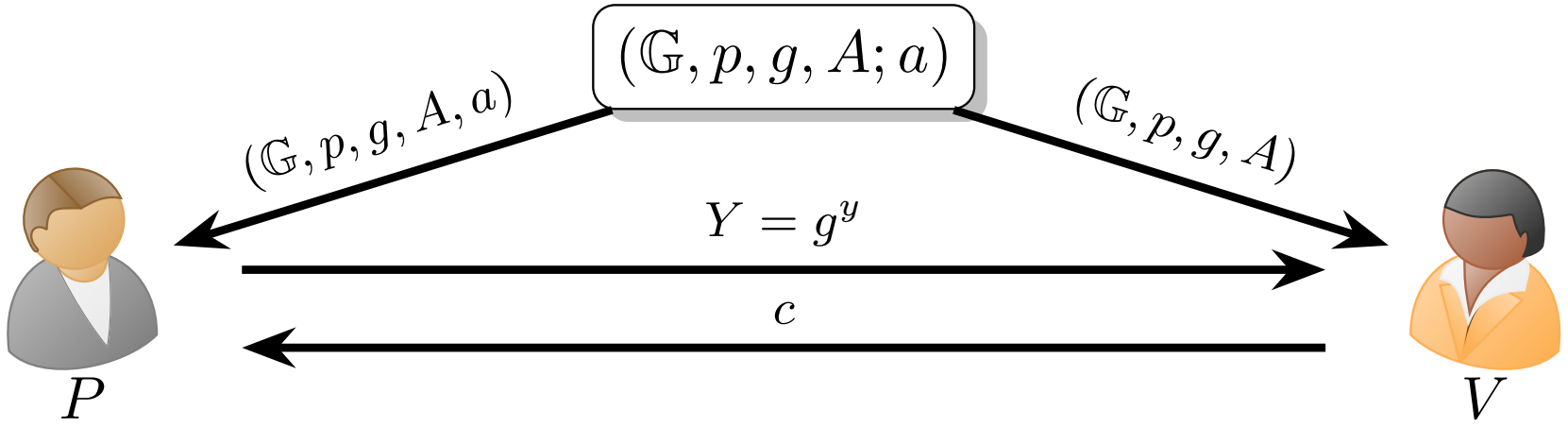
Schnorr Identification Protocol



(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

Schnorr Identification Protocol

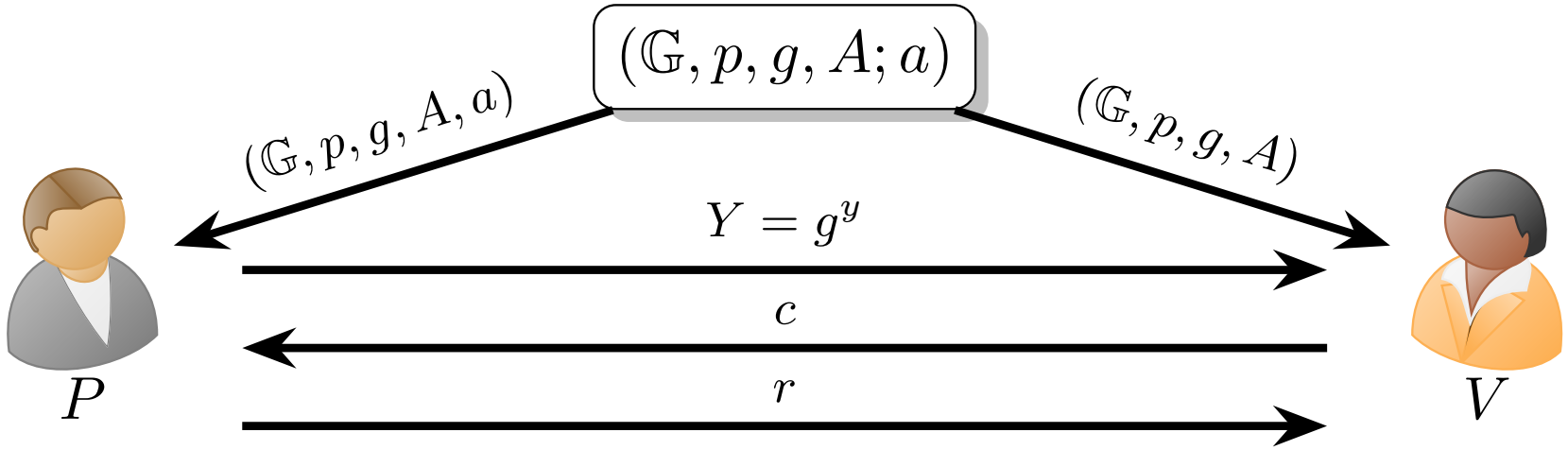


(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(P2) Set
$r := (y + c \cdot a) \bmod p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

Schnorr Identification Protocol

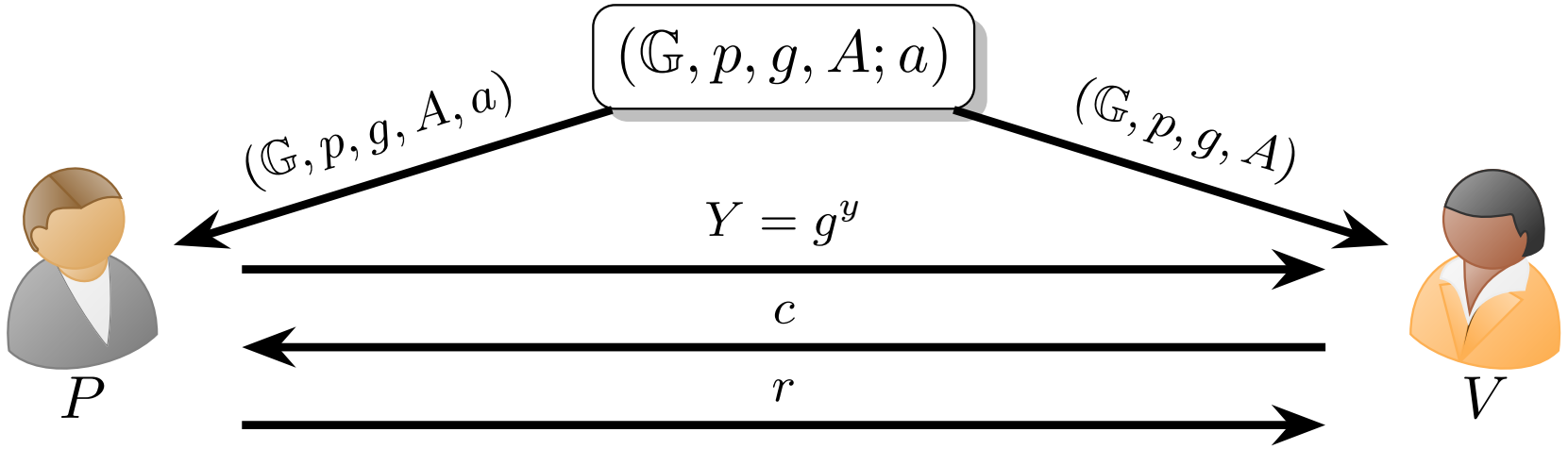


(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(P2) Set
$r := (y + c \cdot a) \bmod p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

Schnorr Identification Protocol



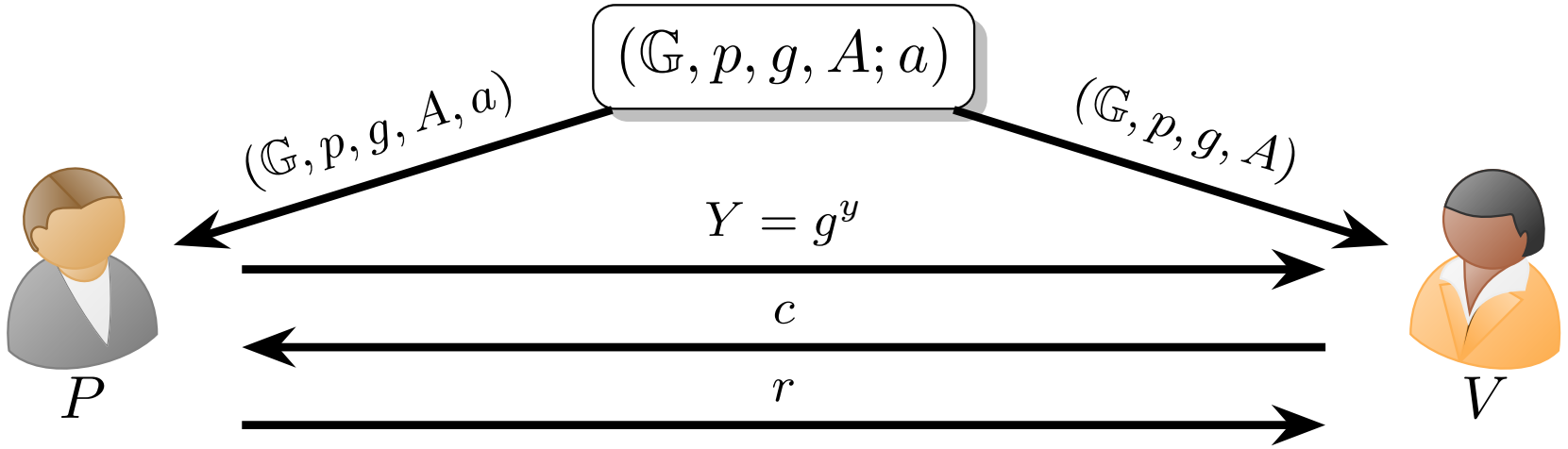
(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(P2) Set
$r := (y + c \cdot a) \bmod p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

(V2) Check if
$g^r = Y \cdot A^c$.

Schnorr Identification Protocol



(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

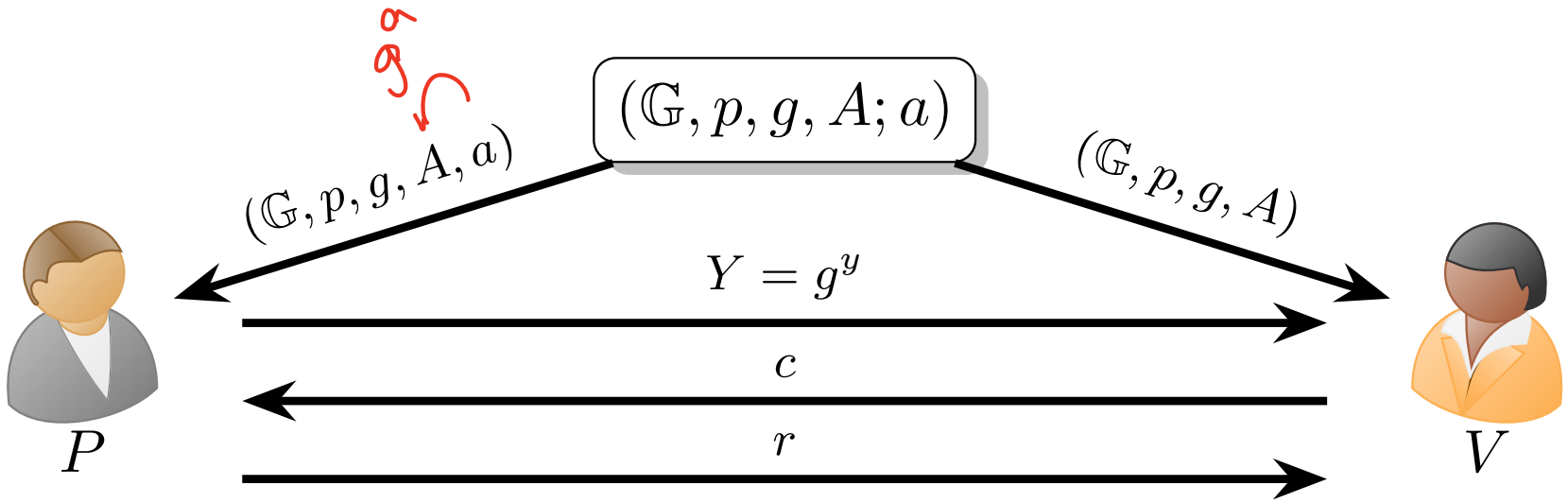
(P2) Set
$r := (y + c \cdot a) \bmod p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

(V2) Check if
$g^r = Y \cdot A^c$.

- Perfect Completeness:

Schnorr Identification Protocol



(P1) Sample $y \xleftarrow{\$} \mathbb{Z}_p$.

(P2) Set
$r := (y + c \cdot a) \bmod p$.

(V1) Sample $c \xleftarrow{\$} \mathbb{Z}_p$.

(V2) Check if
$g^r = Y \cdot A^c$.

- Perfect Completeness:
  - $g^r = g^{y+c \cdot a} = g^y \cdot (g^a)^c = Y \cdot A^c$.

Schnorr Identification Protocol

Schnorr Identification Protocol

- (Special) Soundness.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.
  - A transcript is *accepting* if $g^r = Y \cdot A^c$.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.
  - A transcript is *accepting* if $g^r = Y \cdot A^c$.

Claim 1

There is an efficient algorithm for finding the discrete-log of A (namely, a), given any two accepting transcripts (A, Y, c, r) and (A, Y, c', r') for $c \neq c'$.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.
  - A transcript is *accepting* if $g^r = Y \cdot A^c$.

Claim 1

There is an efficient algorithm for finding the discrete-log of A (namely, a), given any two accepting transcripts (A, Y, c, r) and (A, Y, c', r') for $c \neq c'$.

- The above claim establishes that Schnorr's Protocol is actually a *proof of knowledge*.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.
  - A transcript is *accepting* if $g^r = Y \cdot A^c$.

Claim 1

There is an efficient algorithm for finding the discrete-log of A (namely, a), given any two accepting transcripts (A, Y, c, r) and (A, Y, c', r') for $c \neq c'$.

- The above claim establishes that Schnorr's Protocol is actually a *proof of knowledge*.
- In other words, the prover should only be able to produce accepting transcripts *if it knows* a.

Schnorr Identification Protocol

- (Special) Soundness.
  - A *transcript* of the Schnorr protocol is a tuple (A, Y, c, r), where $A, Y \in \mathbb{G}$ and $c, r \in \mathbb{Z}_p$.
  - A transcript is *accepting* if $g^r = Y \cdot A^c$.

Claim 1

There is an efficient algorithm for finding the discrete-log of A (namely, a), given any two accepting transcripts (A, Y, c, r) and (A, Y, c', r') for $c \neq c'$.

- The above claim establishes that Schnorr's Protocol is actually a *proof of knowledge*.

- In other words, the prover should only be able to produce accepting transcripts *if it knows* a.
  - If P could produce these transcripts *without knowing* a, then we'd be able to break discrete-log in polynomial-time!

Special Soundness of Schnorr

- We prove the claim.

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

$$(g^r) \cdot \left(g^{r'}\right)^{-1} = (Y \cdot A^c) \cdot (Y \cdot A^{c'})^{-1}$$

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

$$(g^r) \cdot \left(g^{r'}\right)^{-1} = (Y \cdot A^c) \cdot (Y \cdot A_{c'})^{-1}$$
$$\implies g^{r-r'} = (Y \cdot Y^{-1}) \cdot A^{c-c'} = A^{c-c'} \neq 0$$

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

$$(g^r) \cdot \left(g^{r'}\right)^{-1} = (Y \cdot A^c) \cdot (Y \cdot A_{c'})^{-1}$$
$$\implies g^{r-r'} = (Y \cdot Y^{-1}) \cdot A^{c-c'} = A^{c-c'}$$

- Since $c - c' \neq 0$, $(c - c')^{-1}$ exists modulo p, which gives us:

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

$$(g^r) \cdot \left(g^{r'}\right)^{-1} = (Y \cdot A^c) \cdot (Y \cdot A_{c'})^{-1}$$
$$\implies g^{r-r'} = (Y \cdot Y^{-1}) \cdot A^{c-c'} = A^{c-c'}$$

- Since $c - c' \neq 0$, $(c - c')^{-1}$ exists modulo p, which gives us:

$$g^{(r-r')\cdot(c-c')^{-1}} = A.$$

Special Soundness of Schnorr

- We prove the claim.

Proof of Claim 1.

- If (A, Y, c, r) and (A, Y, c', r') are both accepting, we have

$$g^r = Y \cdot A^c \qquad g^{r'} = Y \cdot A^{c'}.$$

- Using some algebra, we can compute the following.

$$(g^r) \cdot \left(g^{r'}\right)^{-1} = (Y \cdot A^c) \cdot (Y \cdot A_{c'})^{-1}$$
$$\implies g^{r-r'} = (Y \cdot Y^{-1}) \cdot A^{c-c'} = A^{c-c'}$$

- Since $c - c' \neq 0$, $(c - c')^{-1}$ exists modulo p, which gives us:

$$g_{(r-r')\cdot(c-c')^{-1}} = A.$$

- This implies that $(r - r') \cdot (c - c')^{-1}$ is the discrete-log of A, which we have computed in polynomial-time given the two accepting transcripts. □

HVPZK of Schnorr

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

$$A^c \cdot Y = g^r$$

$$A^c (g^r \cdot A^{-c}) = g^r$$

$$\Rightarrow \quad g^r = g^r$$

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

- Notice that the transcript is accepting *even though the discrete-log of A is not known to the simulator.*

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

- Notice that the transcript is accepting *even though the discrete-log of A is not known to the simulator.*
- c above is identically distributed to c in the real protocol.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

- Notice that the transcript is accepting *even though the discrete-log of A is not known to the simulator.*
- c above is identically distributed to c in the real protocol.
  - r *is also* identically distributed since in the real protocol, $r := y + c \cdot a \bmod p$, which is uniformly in $\mathbb{Z}_p$ since $y \xleftarrow{\$} \mathbb{Z}_p$.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

- Notice that the transcript is accepting *even though the discrete-log of A is not known to the simulator.*
- c above is identically distributed to c in the real protocol.
  - r *is also* identically distributed since in the real protocol, $r := y + c \cdot a \bmod p$, which is uniformly in $\mathbb{Z}_p$ since $y \xleftarrow{\$} \mathbb{Z}_p$.
- This implies that Y from the simulator is also identically distributed to the real protocol Y.

HVPZK of Schnorr

- We prove Schnorr's scheme is honest-verifier PZK.

$S(\mathbb{G}, p, g, A)$:

- Sample $c \xleftarrow{\$} \mathbb{Z}_p$ and $r \xleftarrow{\$} \mathbb{Z}_p$.
- Set $Y := g^r \cdot (A^c)^{-1}$.
- Output (A, Y, c, r).

- Notice that the transcript is accepting *even though the discrete-log of A is not known to the simulator.*
- c above is identically distributed to c in the real protocol.
  - r *is also* identically distributed since in the real protocol, $r := y + c \cdot a \bmod p$, which is uniformly in $\mathbb{Z}_p$ since $y \xleftarrow{\$} \mathbb{Z}_p$.
- This implies that Y from the simulator is also identically distributed to the real protocol Y.
- So the two output distributions are identical.

Does Special Soundness Violate Zero-knowledge?

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.

- However, this implies that V would be able to send c to P, get a response r, then say "ok let's do it again" by sending a different c'.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.

- However, this implies that V would be able to send c to P, get a response r, then say "ok let's do it again" by sending a different c'.
  - In the real-world, P would abort the protocol if V attempted such behavior.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.

- However, this implies that V would be able to send c to P, get a response r, then say "ok let's do it again" by sending a different c'.
  - In the real-world, P would abort the protocol if V attempted such behavior.
  - More likely, the whole protocol would need to be restarted.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.
- However, this implies that V would be able to send c to P, get a response r, then say "ok let's do it again" by sending a different c'.
  - In the real-world, P would abort the protocol if V attempted such behavior.
  - More likely, the whole protocol would need to be restarted.
- So special soundness does not violate zero-knowledge, as V would need to interact with P at least twice with the first same message Y.

Does Special Soundness Violate Zero-knowledge?

- Special soundness implies that if V were given *rewinding access* to P, then the protocol *would not be zero-knowledge*.
  - V just runs the attack we outlined, finds the discrete-log of A.

- However, this implies that V would be able to send c to P, get a response r, then say "ok let's do it again" by sending a different c'.
  - In the real-world, P would abort the protocol if V attempted such behavior.
  - More likely, the whole protocol would need to be restarted.

- So special soundness does not violate zero-knowledge, as V would need to interact with P at least twice with the first same message Y.

  - p is chosen to be cryptographically large (i.e., $\log(p) = \lambda$), so the probability of getting the same first message Y in two independent executions of the protocol is $2^{-\lambda}$.

Next Time: Zero-knowledge for all of NP