Zero Knowledge

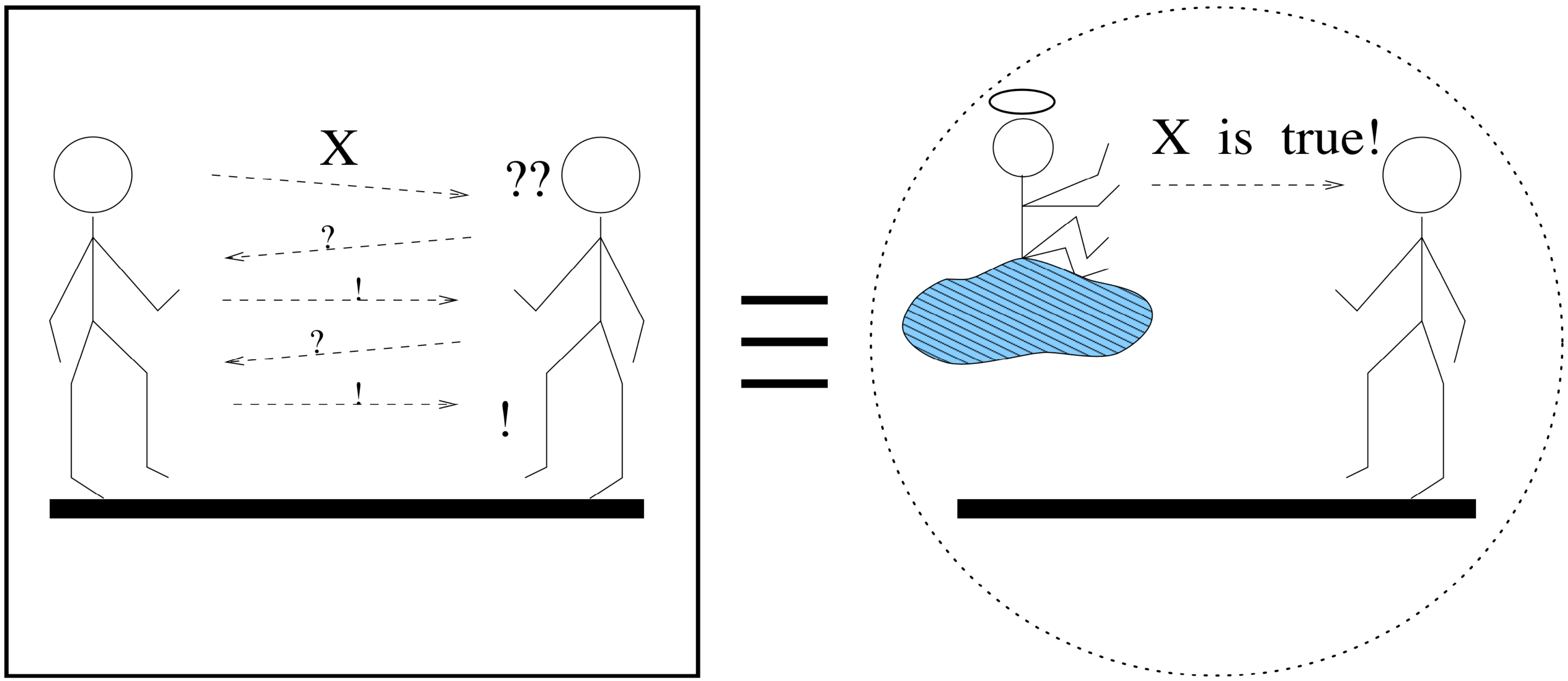
Brice Minaud

email: brice.minaud@inria.fr
website: www.di.ens.fr/brice.minaud/

MPRI, 2019

Zero Knowledge

Goldwasser, Micali, Rackoff '85.



A zero-knowledge course would be a very bad course.

Image credit Oded Goldreich www.wisdom.weizmann.ac.il/~oded/PS/zk-tut10.ps

Expressivity

Zero-knowledge (ZK) proofs are very powerful and versatile.

On an intuitive level (for now), statements you may want to prove:

- "I followed the protocol honestly." (but want to hide the secret values involved.) *E.g. prove election result is correct, without revealing votes.*
- "I know this secret information." (but don't want to reveal it.) *For identification purposes.*
- "The amount of money going into this transaction is equal to the amount of money coming out." (but want to hide the amount, and how it was divided.)

What do we want to prove?

Want to prove a statement on some *x*: P(x) is true.

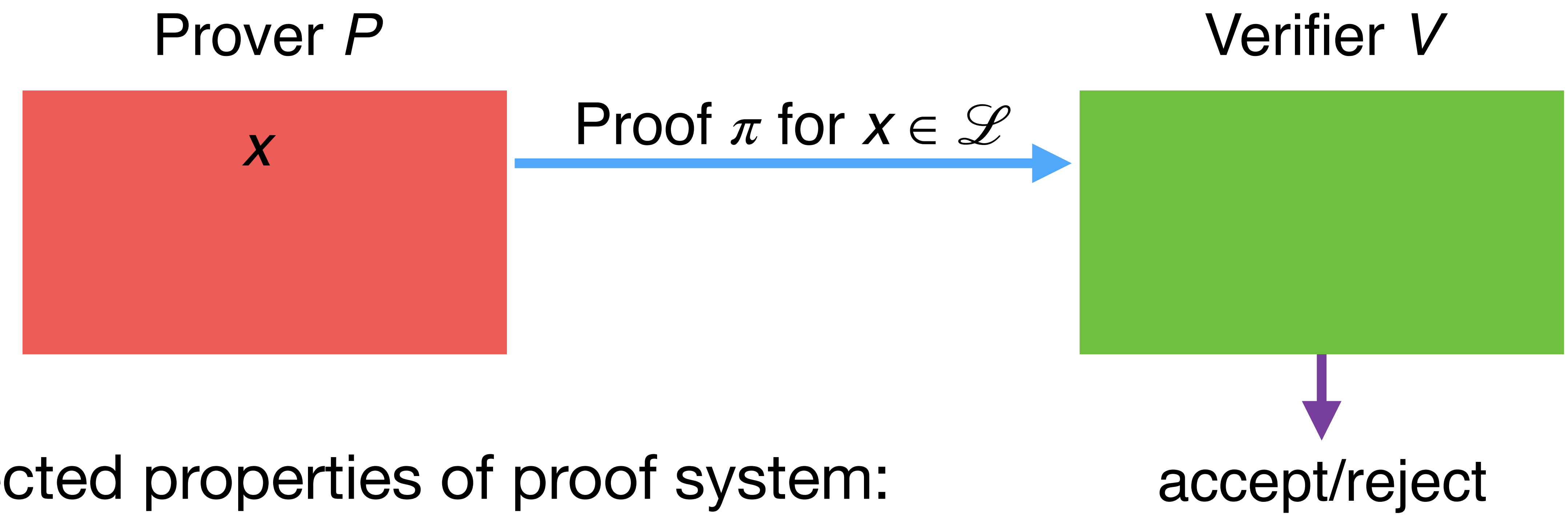
Exemple: x = list V of encryptions of all votes + election result R
P(V,R) = result R is the majority vote among encrypted votes V.

In general, can regard x as a bit string.

Equivalently: want to prove $x \in \mathscr{L}$. (set $\mathscr{L} = \{y : P(y)\}$.)

What is a proof?

For a language $\mathscr{L}$:



Expected properties of proof system:

- Completeness. If $x \in \mathscr{L}$, then $\exists$ proof π, $V(\pi)$ = accept.
- Soundness. If $x \notin \mathscr{L}$, then $\forall$ proof π, $V(\pi)$ = reject.
- Efficiency. V is PPT (Probabilistic Polynomial Time).

Without the last condition, definition is vacuous (prover is useless).

Zero knowledge

Intuitively: Verifier learns *nothing* from π other than $x \in \mathscr{L}$.

...this is impossible for previous notion of proof.

(only possible languages are those in BPP, i.e. when the proof is useless...)

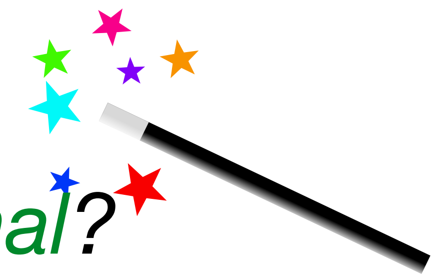
→ going to generalize/relax notion of proofs in a few ways:

- Interactive proof, probabilistic prover, imperfect (statistical) soundness...

Brief interlude: crypto magic

Challenge:

Define an injective mapping F: $\{0,1\}^* \to \{0,1\}^\lambda$.



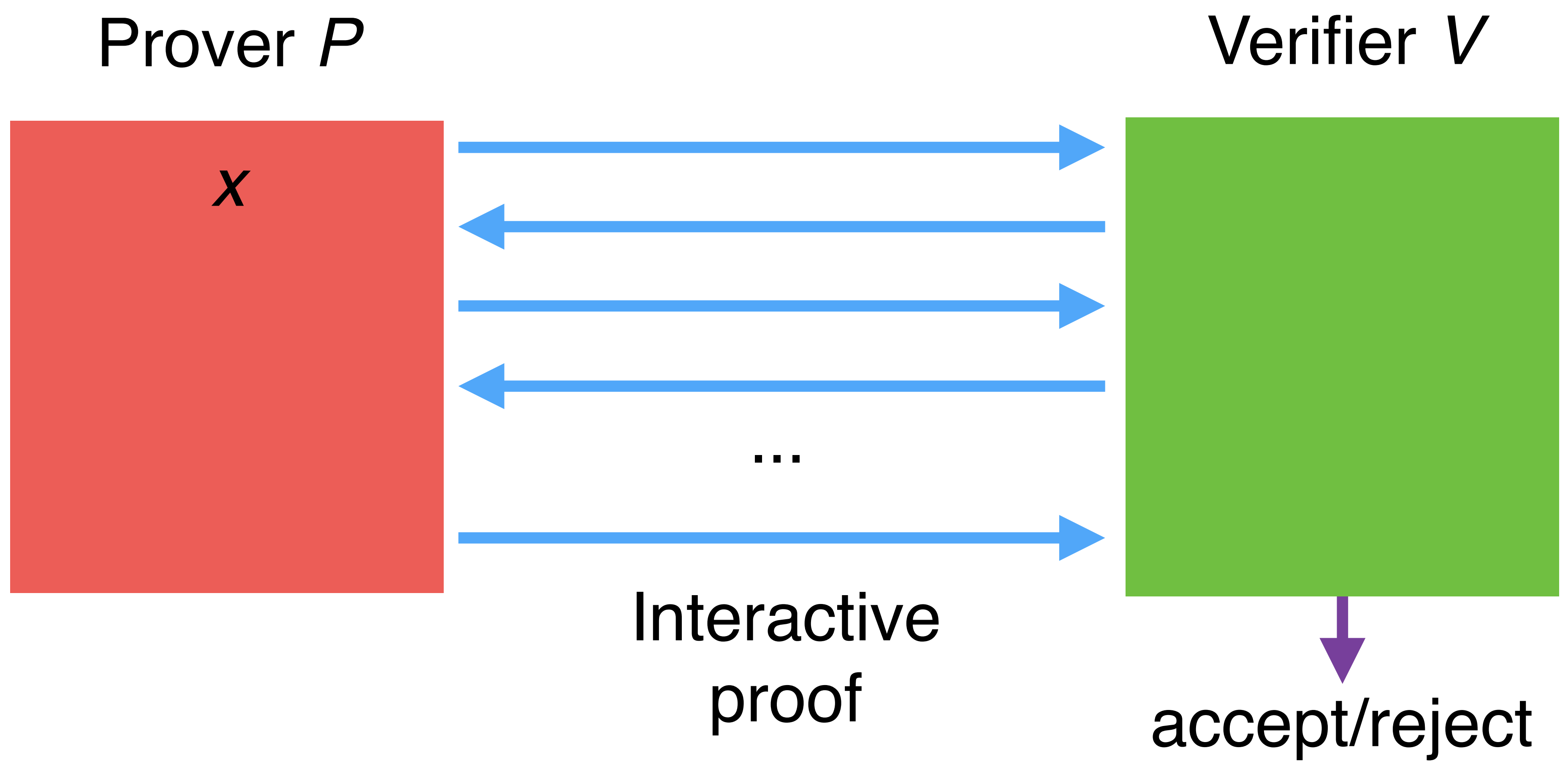
How about if injectivity is only computational?

i.e. computationally hard to find $x \neq y$ s.t. $F(x) = F(y)$.

Then it's fine! It's a (cryptographic) hash function.

(Story for another time: hardness as sketched above is ill-defined.)

Interactive proof



An Interactive Proof (P,V) for $\mathscr{L}$ must satisfy:

- (Perfect) Completeness. If $x \in \mathscr{L}$, then $P \leftrightarrow V$ accepts.
- (Statistical) Soundness. If $x \notin \mathscr{L}$, then $\forall$ prover P^*, $\Pr[P^* \leftrightarrow V \text{ rejects}] =$ non-negl($|x|$). (i.e. $\geq 1/p(|x|)$ for some fixed polynomial p.)
- Efficiency. V is PPT.

Caveat: prover is unbounded.

IP

IP: complexity class of languages that admit an interactive proof.

Public-coin proof: verifier gives its randomness to prover.
Private-coin proof: no such restriction. No more expressive.

Theorem. Shamir, LKFN at FOCS '90.

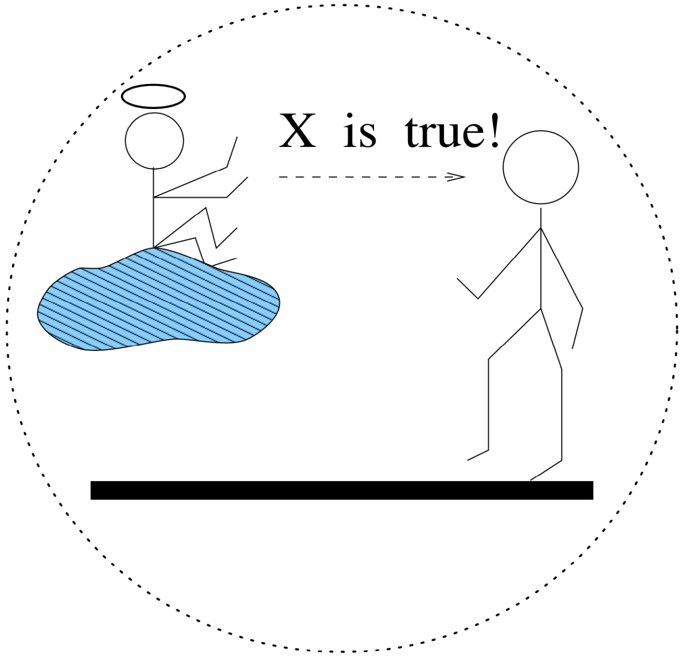
IP = PSPACE.

Very powerful but in crypto, for usability, we want efficient (PPT) prover.

when soundess is wrt PPT prover, sometimes say **argument** of knowledge.

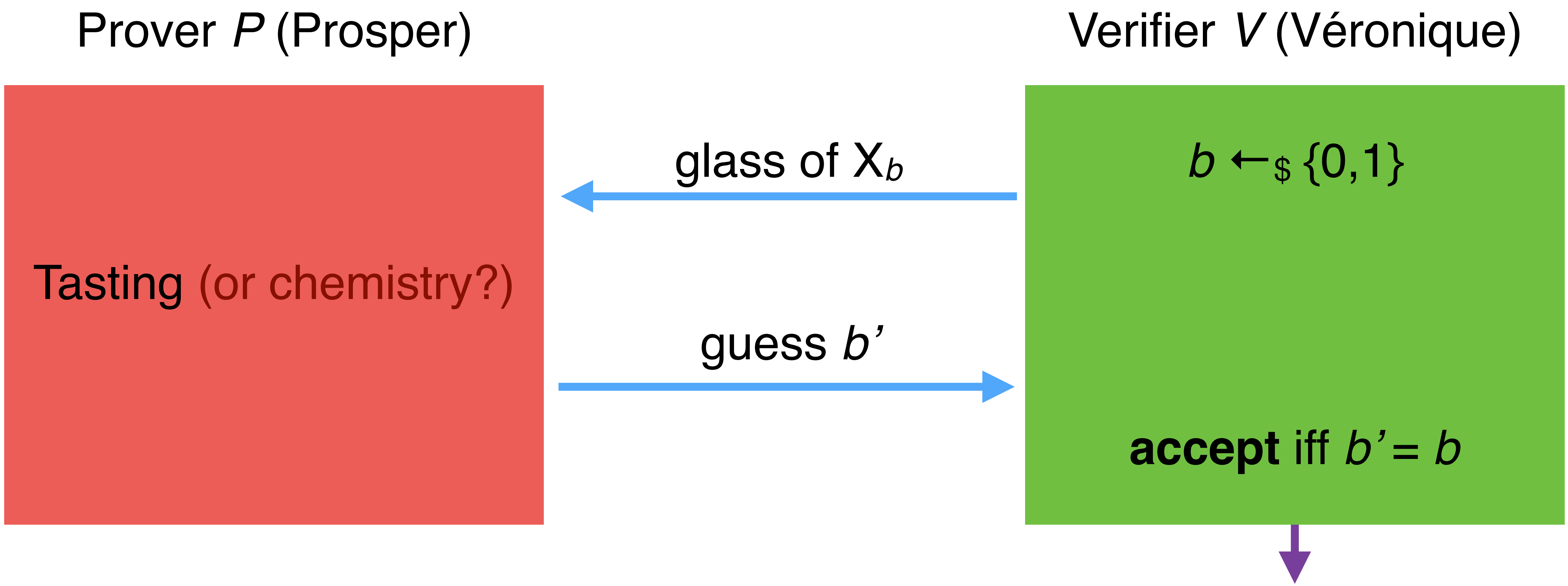
Further, we often want **zero knowledge**.

Zero knowledge



Pepsi or Coke is in IP

Prosper (P) wants to prove to Véronique (V) that she can distinguish Pepsi from Coke. Let (X_0,X_1) = (Pepsi,Coke).



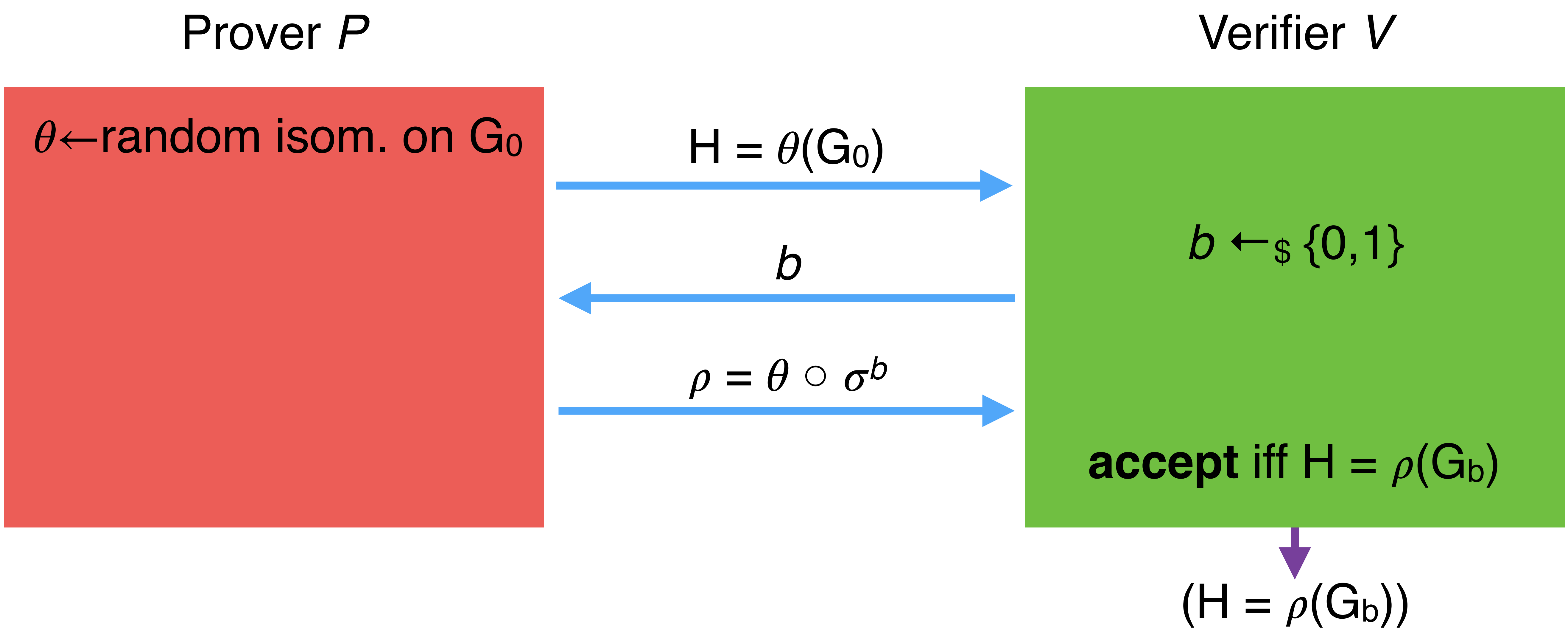
This interactive proof is **complete** and **sound**.

Soundess error = 1/2. Reduce to $2^{-\lambda}$: iterate the protocol λ times.

Graph isomorphism

- I know an isomorphism σ between two graphs G_0, G_1: $\sigma(G_0) = G_1$.
- I want to prove $G_0 \sim G_1$ without revealing anything about the isomorphism.

Formally: $\mathscr{L} = \{(G,G'): G \sim G'\}$, want to prove $(G_0,G_1) \in \mathscr{L}$.



Bounded prover who knows a *witness*. Public coin. Perfect ZK.

Analysis

- (Perfect) Completeness.
  "If $x \in \mathscr{L}$, then $P \leftrightarrow V$ accepts".

  Clearly true.

- (Statistical) Soundness.
  "If $x \notin \mathscr{L}$, then $\forall$ prover P^*, $\Pr[P^* \leftrightarrow V \text{ rejects}] = \text{non-negl}(|x|)$".

  True: V will reject with probability $\geq 1/2$.

- Efficiency. V is PPT.

Analysis

We want to actually use this → want a bounded prover (PPT).

Graph isomorphism: bounded prover is OK if they know a witness: the permutation σ. Note: secret knowledge necessary for bounded prover to make sense.

→ NP languages are great: $\mathscr{L} = \{x \mid \exists w, R(x,w)\}$ for efficient R.

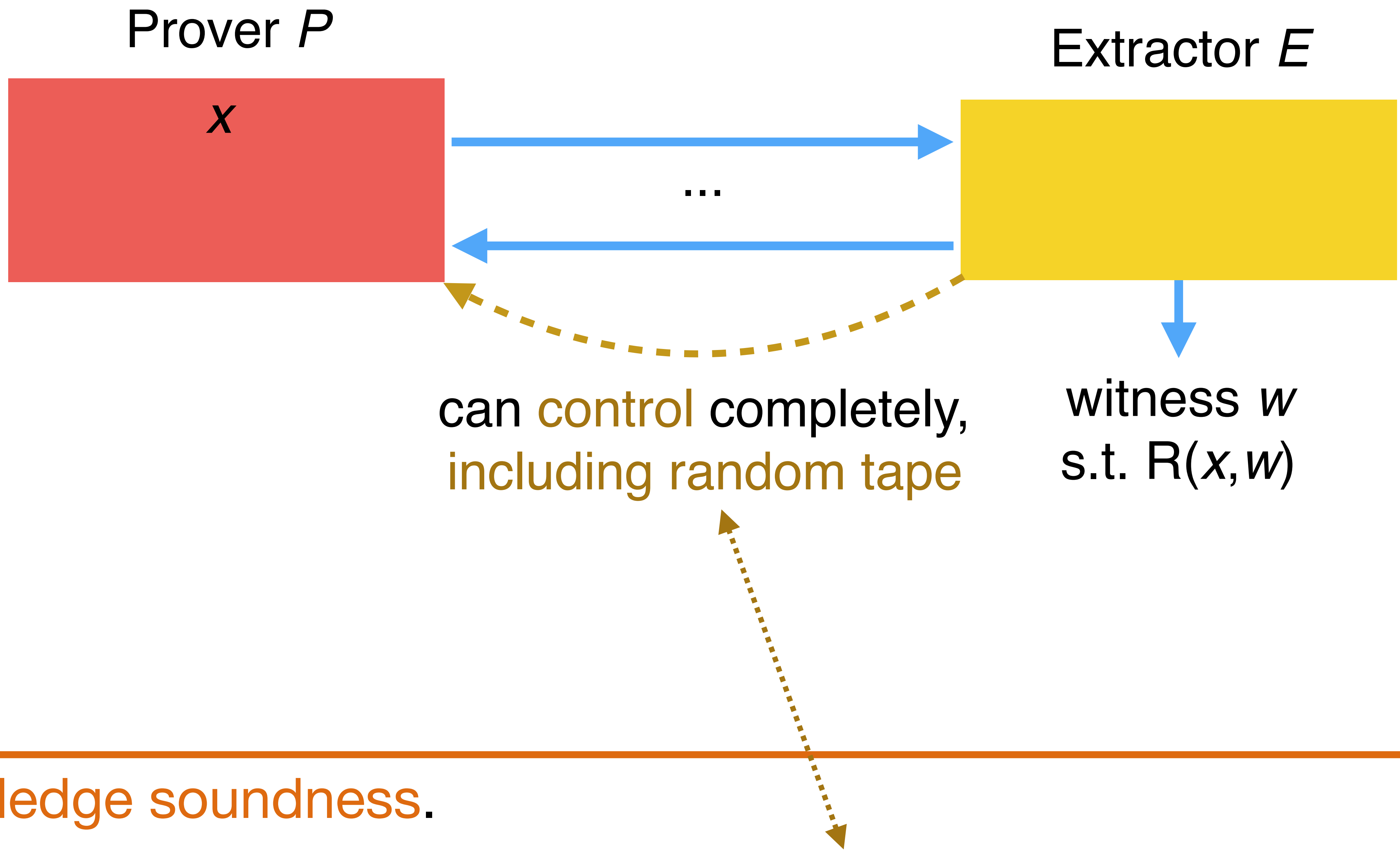
Two proof goals:

- Proof of membership. Want to prove: "$x \in \mathscr{L}$".
- Proof of knowledge. Want to prove: "I know w s.t. $R(x,w)$"

Completeness: unchanged.

Soundness: for membership: already seen. For knowledge: how do you express: "proof implies P 'knows' w"?

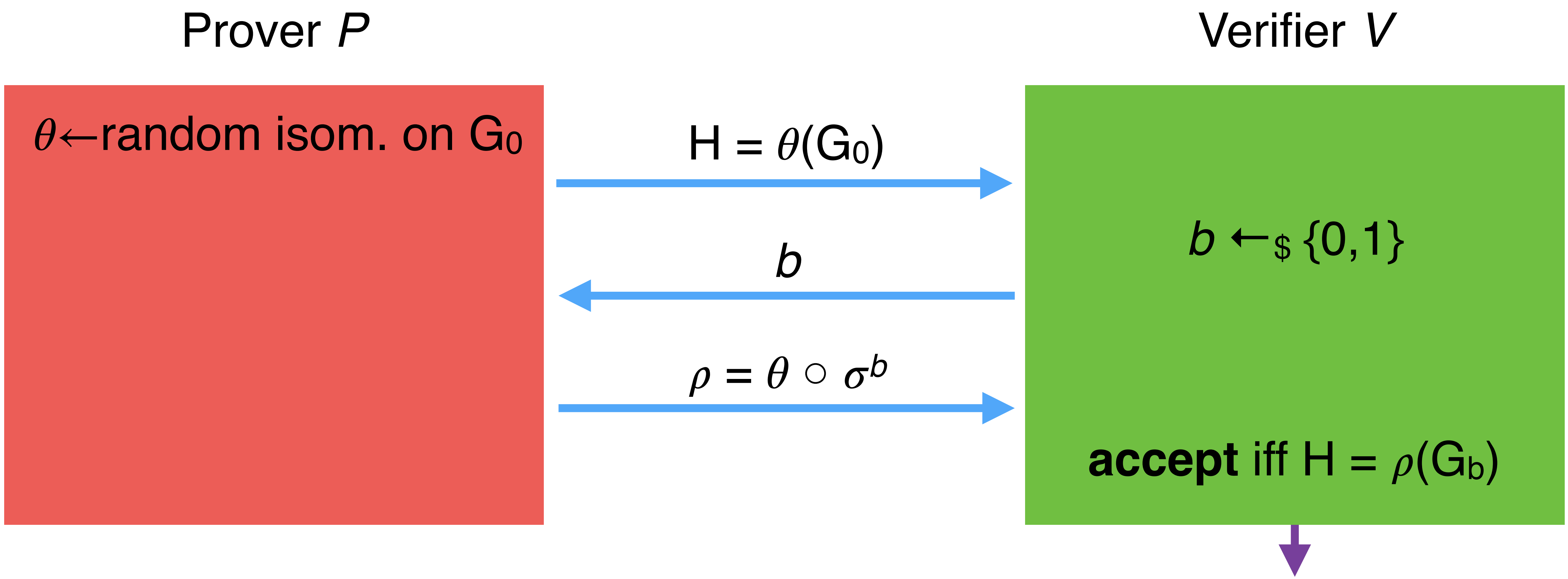
Soundness of a **knowledge** proof



Knowledge soundness.

∃ efficient extractor *E* that, given access to *P* and *x*, can compute *w* such that R(*x*,*w*) (with non-negligible probability, and for any *P* that convinces *V* with non-negligible probability).

Knowledge soundness for Graph Isomorphism



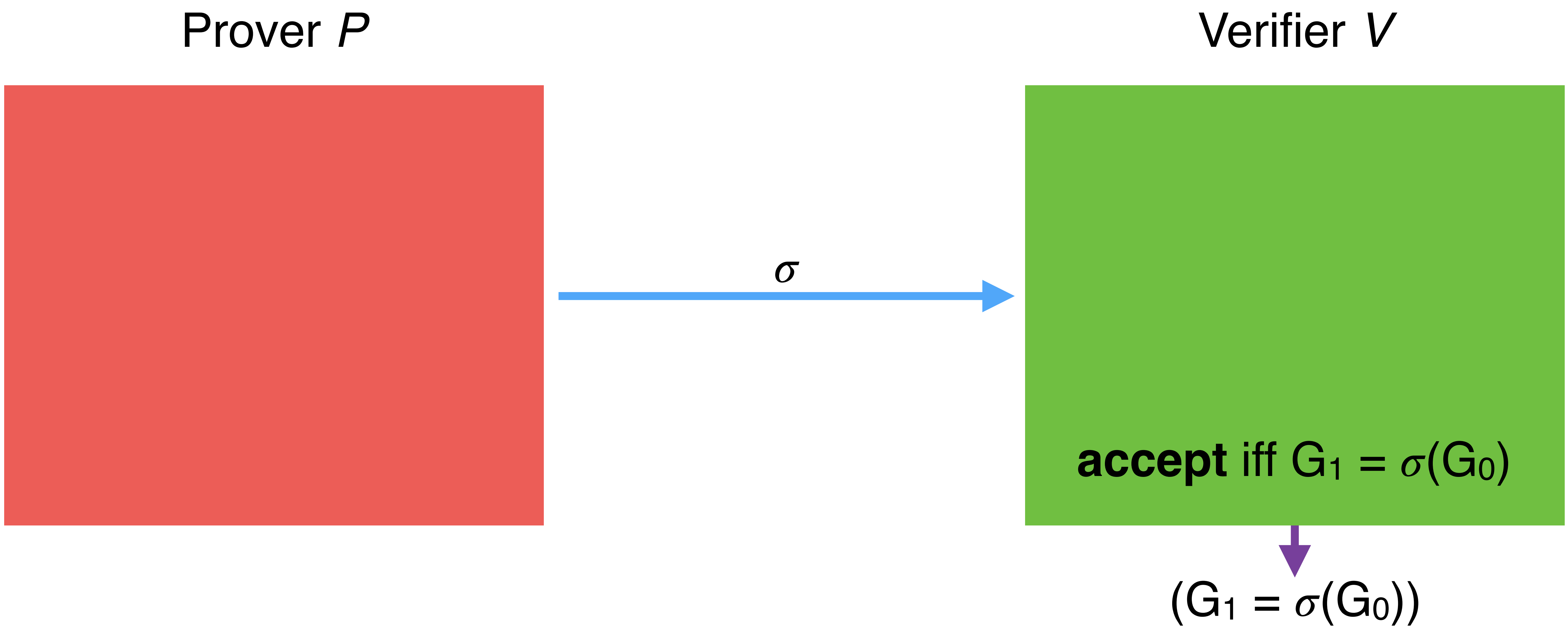
$(H = \rho(G_b))$

Extractor:

- calls P, gets $H = \theta(G_0)$.
- asks $b = 0$, **and** $b = 1$. This is legitimate due to randomness control! Gets back ρ_0, ρ_1 with $H = \rho_0(G_0) = \rho_1(G_1)$.
- $G_1 = \rho_1^{-1} \circ \rho_0(G_0) \rightarrow$ witness $\sigma = \rho_1^{-1} \circ \rho_0$.

Special soundness: answering two challenges reveals witness.

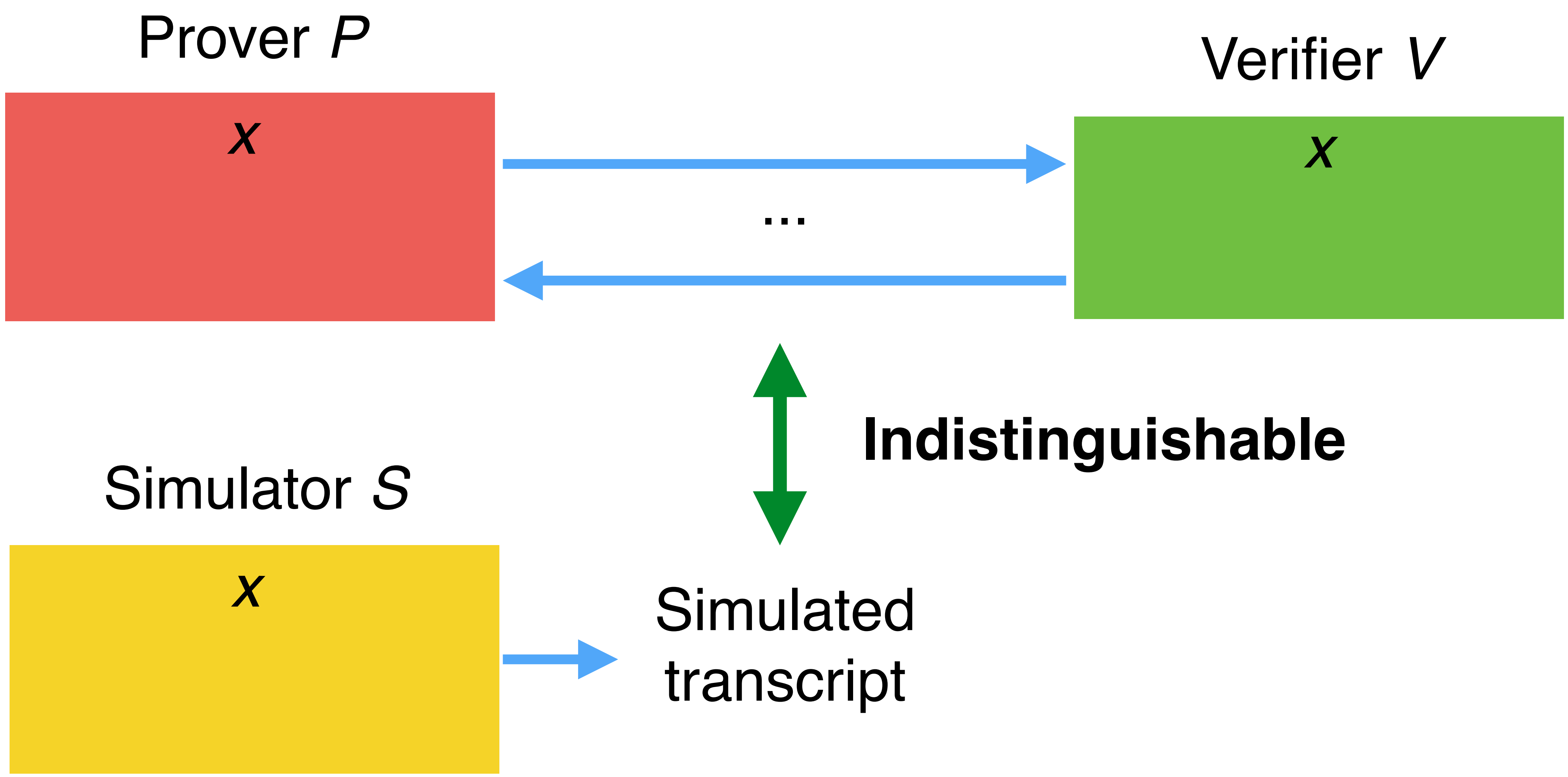
Towards zero knowledge



For language in NP, witness itself *is* a proof of knowledge...

- Zero-knowledge: prove membership or knowledge while revealing *nothing else*.

Honest-verifier zero-knowledge



Honest-verifier zero-knowledge.

The (interactive) proof system (*P*,*V*) is **zero-knowledge** iff:

$\exists$ efficient (PPT) simulator *S* s.t. $\forall\, x \in \mathscr{L}$, transcript of *P* interacting with *V* on input *x* is indistinguishable from the output of *S*(x).

Analysis

Point of definition:

- anything *V* could learn from interacting (honestly) with *P*, could also learn by just running *S*.
- *S* is efficient and knows no secret information.

⇒ Anything *V* can compute with access to *P*, can compute without *P*.

That expresses formally: "*V* learns nothing from *P*".

- Is the Graph Isomorphism proof ZK?

Yes. Simulator: choose b in {0,1}, and random permutation π of G_b.

Publish as simulated transcript: ($\pi(G_b)$,b, π). This is identically distributed to a real transcript → perfect zero-knowledge.

Key argument: $\pi(G_b)$ for uniform π does not depend on b.

Types of zero knowledge

Let ρ be the distribution of real transcrpits, σ simulated transcript.

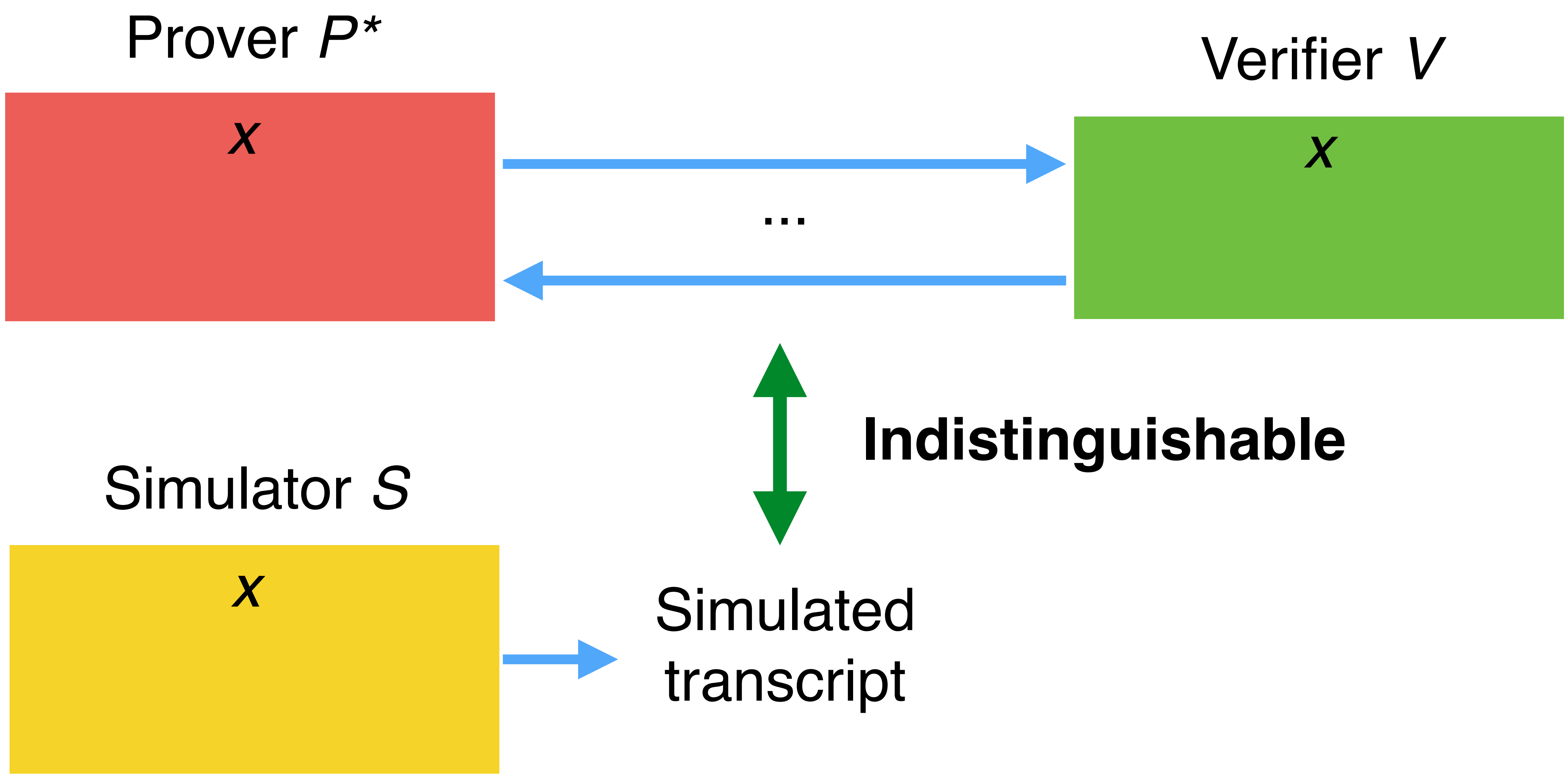
- Perfect ZK: $\rho = \sigma$.
- Statistical ZK: dist(ρ,σ) is negligible. (dist = statistical distance)
- Computational ZK: advantage of efficient adversary trying to distinguish ρ from σ is negligible.

implies

Likewise: completeness, soundness can be perfect/statistical/computational.

What if the prover is **malicious** (does not follow the protocol?)

~~Honest-verifier~~ Zero-knowledge



Zero-knowledge.

The (interactive) proof system (P,V) is **zero-knowledge** iff:

$\forall$ prover P^*, $\exists$ PPT simulator S s.t. $\forall\, x \in \mathscr{L}$, transcript of P^* interacting with V on input x is indistinguishable from output of S(x).

Summary

A ZK proof is (perfectly/statistically/computationally):

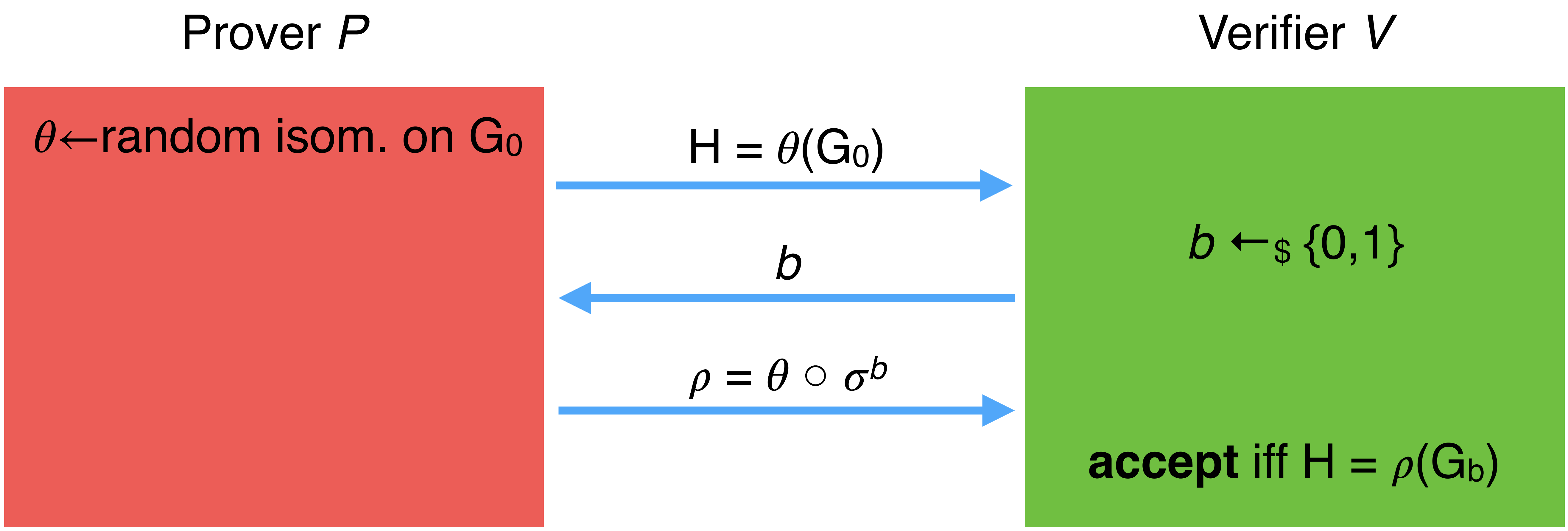
1. Complete
2. Sound
3. Zero-knowledge.

Examples

Graph isomorphism

- I know an isomorphism σ between two graphs G_0, G_1: $\sigma(G_0) = G_1$.
- I want to prove $G_0 \sim G_1$ without revealing anything about the isomorphism.

Formally: $\mathscr{L} = \{(G,G'): G \sim G'\}$, want to prove $(G_0,G_1) \in \mathscr{L}$.

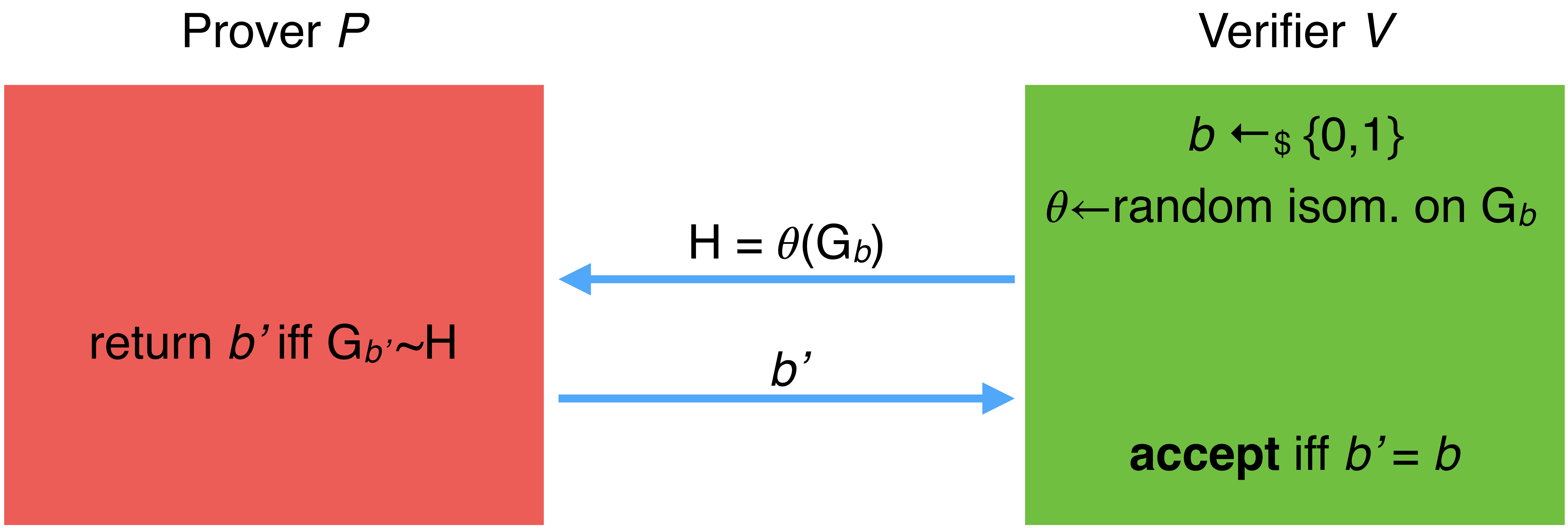


Bounded prover who knows a *witness*. Public coin. Perfect ZK.

Graph **non** isomorphism

- I am an unbounded prover who knows $G_0 \nsim G_1$.
- I want to prove $G_0 \nsim G_1$ without revealing anything else.

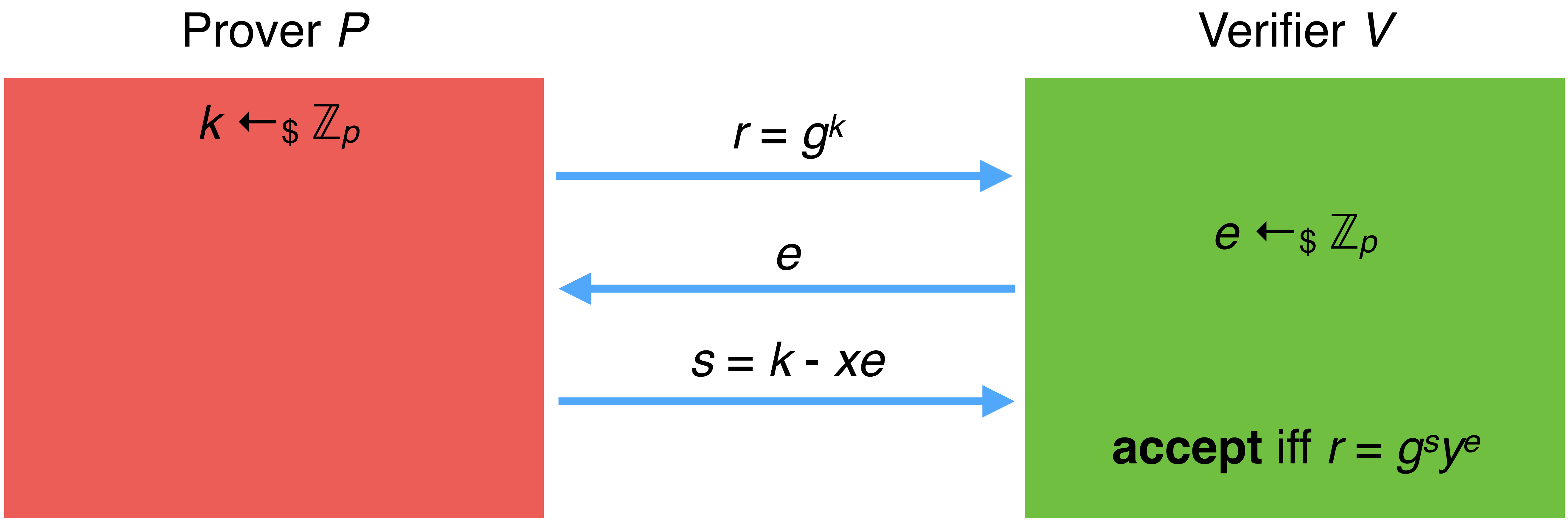
Formally: $\mathscr{L} = \{(G,G'): G \nsim G'\}$, want to prove $(G_0,G_1) \in \mathscr{L}$.



Unbounded prover. Private coin. Not ZK for malicious *V*. Hints IP≠NP.

Knowledge of a discrete log

- Let $\mathbb{G} = \langle g \rangle \sim \mathbb{Z}_p$ and $y \in \mathbb{G}$. I know $x \in \mathbb{Z}_p$ such that $y = g^x$.
- Corresponding language is trivial! $\forall y\ \exists x,\ y = g^x$. But proof of knowledge still makes sense.



Known as Schnorr protocol.

Analysis of Schnorr protocol

- (Perfect) Completeness.

  Clear.

- (Special) Knowledge soundness.

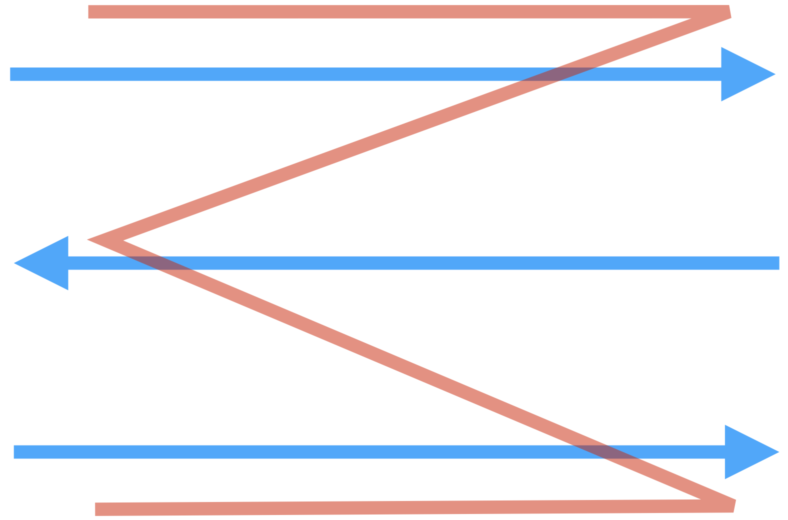
  Extractor: gets $r = g^k$, asks two challenges $e \neq e'$, gets back s, s' with $r = g^s y^e = g^{s'} y^{e'}$. Yields $y = g^{(s-s')/(e'-e)}$.

- (Perfect) Honest-verifier zero knowledge.

  Simulator: draw $e \leftarrow_\$ \mathbb{Z}_p$, $s \leftarrow_\$ \mathbb{Z}_p$, **then** $r = g^s y^e$. Return transcript (r,e,s). Note r, e still uniform and independent → distribution is identical to real transcript.

We will use this for a signature!

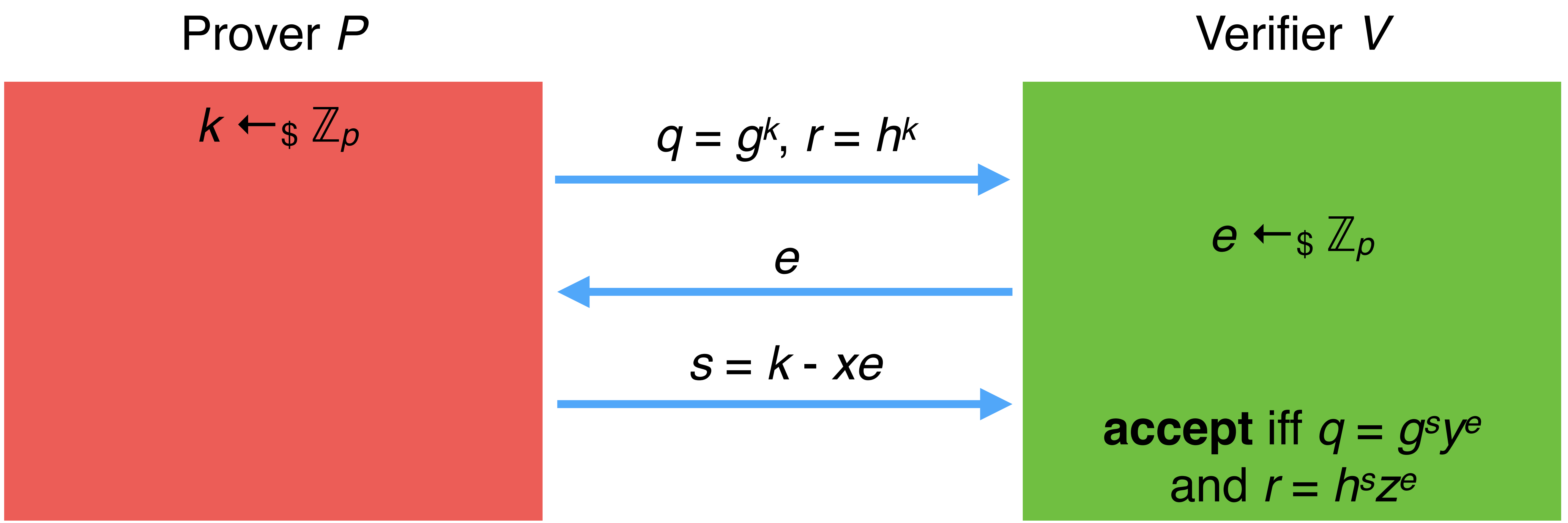
Sigma protocols and NIZK



Equality of exponents = DH language

- Let $\mathbb{G} \sim \mathbb{Z}_p$, $g, h \in \mathbb{G}$. I know $x \in \mathbb{Z}_p$ such that $(y, z) = (g^x, h^x)$.
- Corresponding language is Diffie-Hellman language (for fixed g, h)!

$$\mathscr{L} = \{(g, g^a, g^b, g^{ab}) \colon a, b \in \mathbb{Z}_p\} \leftrightarrow \mathscr{L}' = \{(g^a, h^a) \colon a \in \mathbb{Z}_p\} \text{ for } h = g^b$$

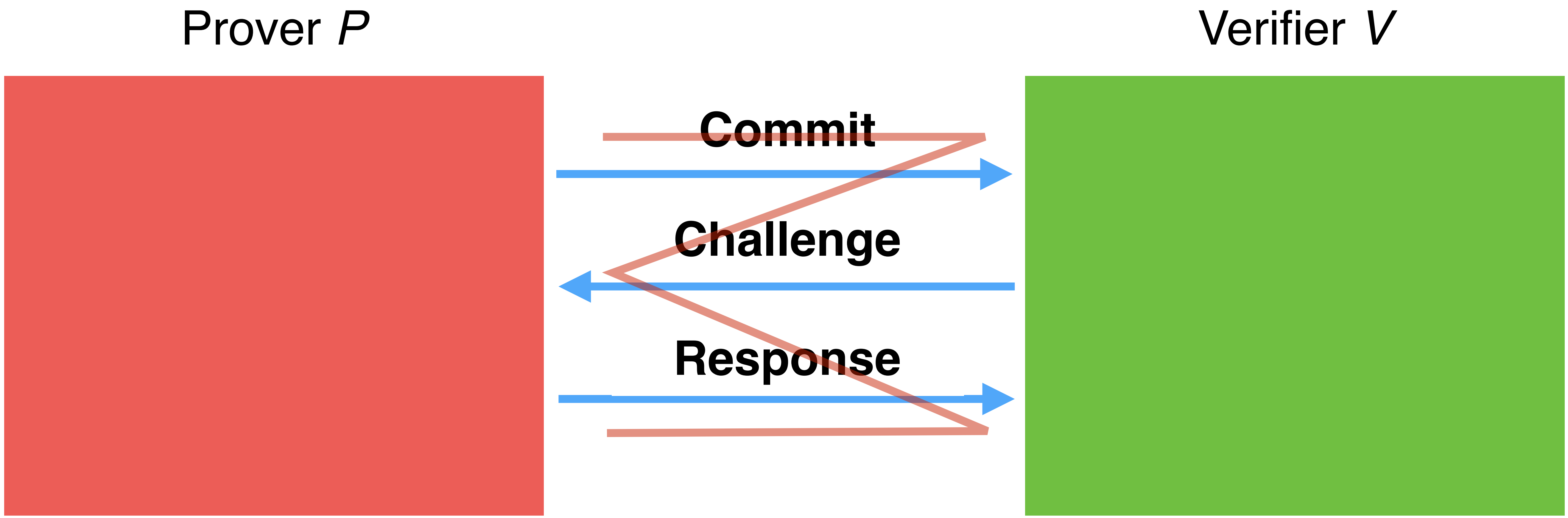


This is two 'simultaneous' executions of Schnorr protocol, with same (k,e). Soundness and ZK proofs are the same.

We will use this in a voting protocol!

Sigma protocol

Schnorr protocol:



Public-coin ZK protocols following this pattern = Sigma Protocols.

Fiat-Shamir transform:

By setting **Challenge** = Hash(**Commit**), can be made non-interactive
→ Non-Interactive Zero-Knowledge (NIZK)

Sigma protocol → signature

NIZK knowledge proof: "I know a witness *w* for R(*x*,*w*)" and can prove it non-interactively without revealing anything about *w*.

This is an identification scheme.

Sigma protocol → can integrate message into challenge randomness.

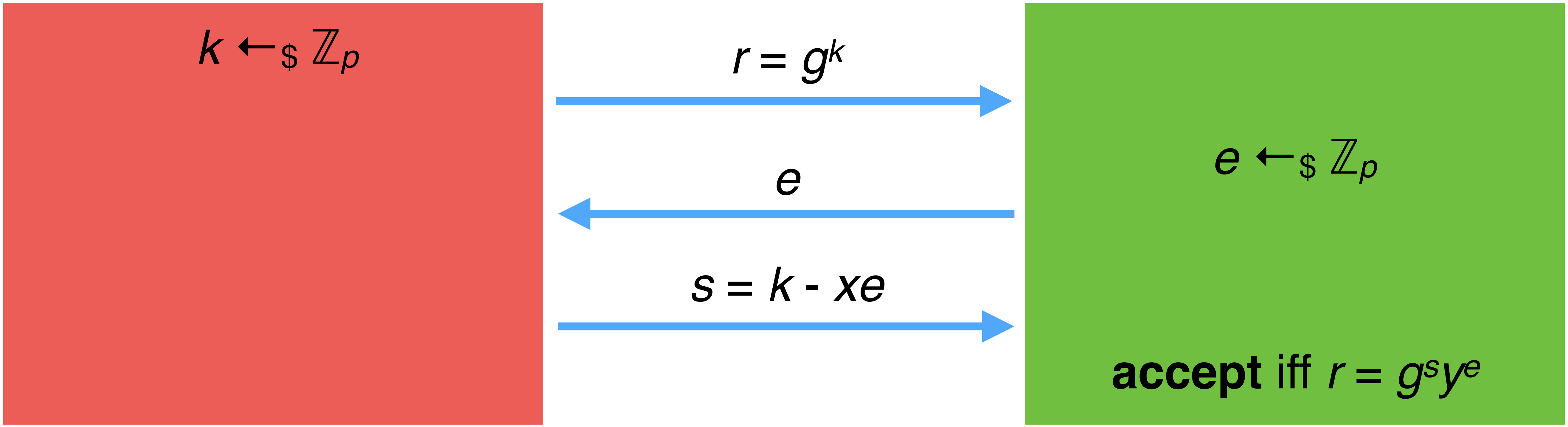
This yields a signature scheme!

- **Public key**: *x*
- **Secret key**: *w*
- **Sign**(*m*): signature = NIZK proof with challenge = hash(commit,*m*)
- **Verify** signature = verify proof.

That is the Fiat-Shamir transform.

Example: Schnorr signature

Schnorr protocol:



Schnorr signature:

Public key: $y = g^x$

Secret key: x

Sign(m): signature $\sigma = (r,s)$ with $r = g^k$ for $k \leftarrow_{\$} \mathbb{Z}_p$, $s = k - xH(r,m)$.

Verify(σ,m): accept iff $r = g^s y^{H(r,m)}$.

Security reduces to Discrete Log, in the Random oracle Model.

ZK proofs for arbitrary circuits

Reductions

Suppose there exists an efficient (polynomial) reduction from $\mathscr{L}$ ' to $\mathscr{L}$:
∃ efficient f such that $x \in \mathscr{L}$ ' iff $f(x) \in \mathscr{L}$. (Karp reduction.)

If I can do ZK proofs for $\mathscr{L}$, I can do ZK proofs for $\mathscr{L}$ '!

To prove $x \in \mathscr{L}$ ', do a ZK proof of $f(x) \in \mathscr{L}$.

Also works for knowledge proofs (via everything being constructive).

⇒ **The dream:** if we can do ZK proof for an NP-complete language, we can prove everything we ever want!

Notably circuit-SAT.

Commitment scheme

A **commitment scheme** is a family of functions C: X x A → V s.t.:

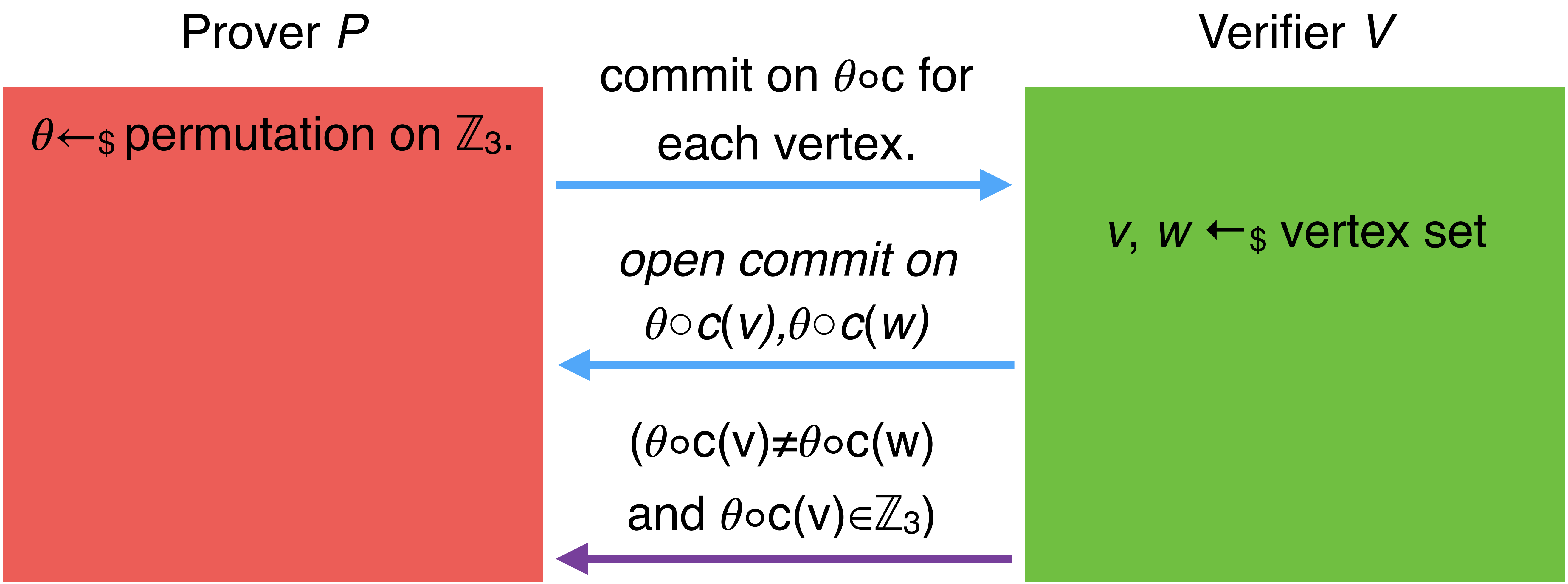
- Binding: it is hard to find x≠x' and a, a' s.t. C(x,a) = C(x',a').
- Hiding: for all x, x', the distributions C(x,a) for a $\leftarrow_\$$ A and C(x',a) for a $\leftarrow_\$$ A are indistinguishable.

Instantiation: pick a hash function.

The dream: ZK proof for 3-coloring

- I know an 3-coloring *c* of a graph G (into $\mathbb{Z}_3$).
- I want to prove that such a coloring exists, without revealing anything about the coloring.

Formally: $\mathscr{L}$ = {(G): G admits a 3-coloring}



Bounded prover with a *witness*. Public coin. Computational ZK.

The wake-up

...this is incredibly inefficient.

- transform circuit-SAT instance into 3-coloring instance.

- run previous protocol *many* times (roughly #circuit size × security parameter) → gigantic proofs, verification times...

SNARKs

SNARK(?) tile by William Morris.

Finite Fields

Most of what follows is going to happen in a finite field.

For a short presentation of finite fields, see:

https://www.di.ens.fr/brice.minaud/cours/ff.pdf

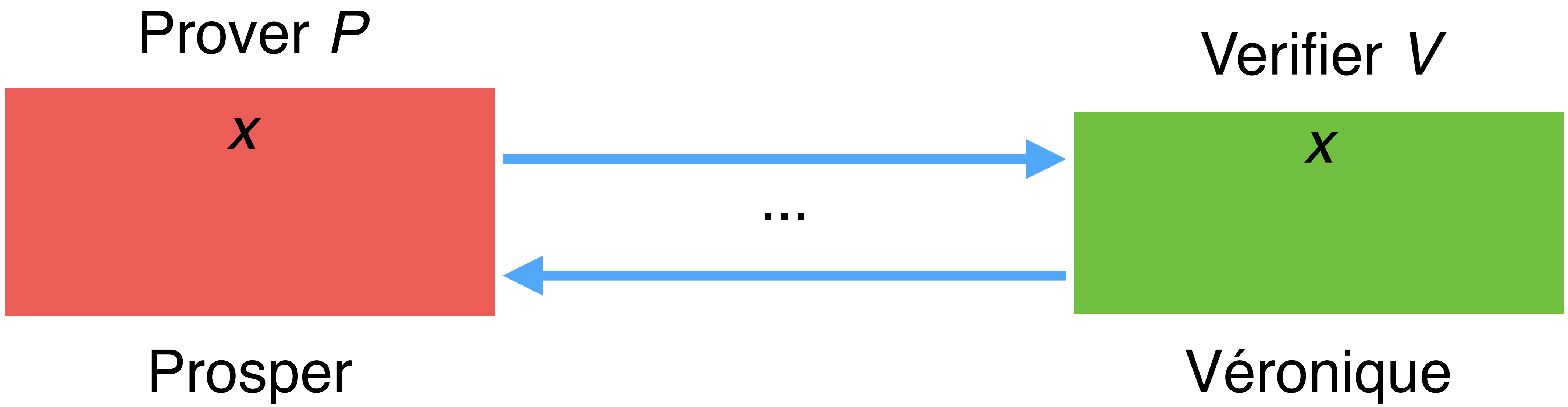
A **key idea** we will use:

If $P \neq Q$ are two degree-d polynomials over $\mathbb{F}_q$, then for $\alpha \leftarrow \mathbb{F}_q$ drawn uniformly at random, $\Pr[\, P(\alpha) \neq Q(\alpha)\,] \geq 1 - d/q$.

→ to check if two bounded-degree polynomials are equal, it is enough to check at a random point!

Proof: P-Q is a non-zero polynomial of degree at most d, so it can be zero on at most d points.

A toy example



Véronique wants to compute the 1000th Fibonacci number in $\mathbb{Z}_p$.

She doesn't have time, so she asks Prosper to to it. But she wants a *proof* that the computation was correct.

“Solution”: agree on whole computation circuit → encode as SAT problem → transform into 3-coloring problem → include NIZK proof of that 3-coloring problem with the result.

Remark: size of proof is linear in the size of the circuit Véronique doesn't want to compute.

(P & V hate closed formulas and fast exponentiation.)

SNARK

We would like to achieve zero-knowledge proofs that are **succint** and non-interactive.

Succint **N**on-interactive **Ar**gument of **K**nowledge: **SNARK**.

Also a fantastical beast by Lewis Caroll:

A new approach

Prosper computes the Fibonacci sequence $f_1, \ldots, f_{1000}$ in $\mathbb{Z}_p$.
He sends f_1, f_2, and f_{1000} to Véronique.

Now V. wants to check $f_{i+2} = f_i + f_{i+1}$ for all i's.

Magic claim: she will be able to check that this computation was correct, for all i, with 99% certainty, by asking Prosper for only 4 values in $\mathbb{Z}_p$.

Disclaimers:
- we assume Prosper answers queries honestly (for now).
- from now on, assume $|\mathbb{Z}_p|$ is "large enough", say $|\mathbb{Z}_p| > 100000$.
  (Otherwise, just go to a field extension.)

This line of presentation is loosely borrowed from Eli Ben-Sasson:
https://www.youtube.com/watch?v=9VuZvdxFZQo

A new approach

Setup: Prosper interpolates a degree-1000 polynomial P in $\mathbb{Z}_p$ such that $P(i) = f_i$ for $i = 1, ..., 1000$.

Let $D = (X-1)\cdot(X-2)\cdot ...\cdot(X-998)$.

$$P(i+2) - P(i+1) - P(i) = 0 \text{ for } i = 1,...,998$$
$$\Rightarrow D \text{ divides } P(X+2) - P(X+1) - P(X)$$
$$\Rightarrow P(X+2) - P(X+1) - P(X) = D\cdot H \text{ for some } H \text{ of degree 2}$$

How Véronique checks that the computation was correct:

- Véronique draws $\alpha \leftarrow \mathbb{Z}_p$ uniformly, computes $D(\alpha)$.
- She asks Prosper for $P(\alpha)$, $P(\alpha+1)$, $P(\alpha+2)$, $H(\alpha)$.
- She accepts computation was correct iff:

$$P(\alpha+2) - P(\alpha+1) - P(\alpha) = D(\alpha)\cdot H(\alpha)$$

Why the approach works

Completeness: if Prosper computed the f_i's correctly, then he can compute $H(\alpha)$ as required.

Soundness: if Prosper computed the f_i's incorrectly, then no matter what degree-two polynomial H Prosper computes:

$$\Pr[\ P(\alpha+2) - P(\alpha+1) - P(\alpha) = D(\alpha) \cdot H(\alpha)\ ] \leq 1000/p < 0.01$$

so Véronique will detect the issue with > 99% probability.

It remains to force Prosper to answer queries honestly.

In particular, soundness argument crucially relies on P, H being bounded-degree polys.

→ need to limit Prosper to computing polys of degree < 1000.

→ A new ingredient: **pairings.**

Pairings

Pairings. Let $\mathbb{G} = \langle g \rangle$, $\mathbb{T} = \langle t \rangle$ be two cyclic groups of order p. A map e: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{T}$ is a *pairing* iff for all a, b in $\mathbb{Z}_p$,

$$e(g^a, g^b) = t^{ab}.$$

Remarks:

- Definition doesn't depend on choice of generators, as long as $t = e(g,g)$.

- Assume Discrete Log is hard in $\mathbb{G}$, otherwise this is useless. On the other hand, e implies DDH cannot be hard (why?).

- First two groups need not be equal in general.

- Can be realized with $\mathbb{G}$ an elliptic curve, $\mathbb{T} = \mathbb{F}_q^*$.

Encodings

Fix $\mathbb{G} = \langle g \rangle$ of order p.

Encode a value $a \in \mathbb{Z}_p$ as g^a. We will write $[a] = g^a$.

We assume DL is hard → decoding a *random* value is hard. But encoding is deterministic → checking if $h \in \mathbb{G}$ encodes a given value is easy.

Additive homomorphism: given encodings $[a]$,$[b]$ of a and b, can compute encoding of $a+b$: $[a+b] = [a][b]$.

→ can compute $\mathbb{Z}_p$-**linear** functions over encodings.

Idea: a pairing $e: \langle g \rangle \times \langle g \rangle \to \langle t \rangle$ allows computing **quadratic** functions over encodings (at the cost of moving to $\mathbb{T}$).

Keeping Prosper honest, using encodings

First: want to ensure P computed by Prosper is degree ≤ 1000.

Approach:

- Véronique draws evaluation point $\alpha \leftarrow \mathbb{Z}_p$ uniformly at random.

- V. publishes encodings $[\alpha]$, $[\alpha^2]$, ..., $[\alpha^{1000}]$.

→ Prosper can compute $[P(\alpha)]$, because it is a linear combination of the $[\alpha^i]$'s, $i \leq 1000$. <u>But only for deg(P) ≤ 1000.</u>
E.g. cannot compute $[\alpha^{1001}]$.

Prosper can compute in the same way $[P(\alpha)]$, $[P(\alpha+1)]$, $[P(\alpha+2)]$, $[H(\alpha)]$.

Remark: Prosper can compute $[(\alpha+1)^i]$ from the $[\alpha^j]$'s for $j \leq i$.

Remaining issues:

1) ensure value “[$P(\alpha)$]”returned by Prosper is in fact a linear combination of [α^i]'s.

2) ensure deg(H) ≤ 2, not 1000.

3) ensure [$P(\alpha)$], [$P(\alpha+1)$], [$P(\alpha+2)$] are from same polynomial.

4) last issue: how does Véronique check the result? Cannot decode encodings.

Dealing with issues (1) and (2)

Goal

1) ensure [$P(\alpha)$] is in fact a linear combination of [α^i]'s.

2) ensure $\deg(H) \leq 2$, not 1000.

Solution:

V. publishes encodings [α], [α^2], ..., [α^{1000}]...

...and also encodings [γ], [$\gamma\alpha$], [$\gamma\alpha^2$], ..., [$\gamma\alpha^{1000}$] for a uniform γ.

→ Prosper can compute [$P(\alpha)$], and [$\gamma P(\alpha)$], and send them to V.

V. can now use the pairing e to check: $e([P(\alpha)],[\gamma]) = e([\gamma P(\alpha)],[1])$.

The point: if Prosper did not compute [$P(\alpha)$] as linear combination of [α^i]'s, he cannot compute [$\gamma P(\alpha)$]. (Note this is quadratic.)

This is an ad-hoc *knowledge assumption* (true in a generic model).

Goal

1) ensure $[P(\alpha)]$ is in fact a linear combination of $[\alpha^i]$'s.

2) ensure deg(*H*) ≤ 2, not 1000.

Solution:

V. publishes encodings $[\alpha]$, $[\alpha^2]$, ..., $[\alpha^{1000}]$...

...and also encodings $[\eta]$, $[\eta\alpha]$, $[\eta\alpha^2]$ for a uniform η.

→ Prosper can compute $[H(\alpha)]$, and $[\eta H(\alpha)]$.

V. can check: $e([H(\alpha)],[\eta]) = e([\eta H(\alpha)],[1])$.

The point: if Prosper did not compute $[H(\alpha)]$ as linear combination of $[\alpha^i]$'s, $i \leq 2$, he cannot compute $[\eta H(\alpha)]$.

Dealing with issue (3)

Goal

3) ensure $[P(\alpha)]$, $[P(\alpha+1)]$, $[P(\alpha+2)]$ are from same polynomial.

Solution:

Let's deal with $[P(\alpha)]$, $[P(\alpha+1)]$.

V. publishes $[\theta]$, $[\theta((\alpha+1)^2-\alpha^2)]$, ..., $[\theta((\alpha+1)^{1000}-\alpha^{1000})]$ for a uniform θ.

→ Prosper can compute $[\theta(P(\alpha+1)-P(\alpha))]$.

V. can check: $e([\theta(P(\alpha+1)-P(\alpha))],[1]) = e([P(\alpha+1)-P(\alpha)],[\theta])$.

The point: if Prosper did not compute $[P(\alpha)]$, $[P(\alpha+1)]$ with same coefficients, he cannot compute $[\theta(P(\alpha+1)-P(\alpha))]$.

Checking divisibility

Summary of 3 previous slides: we have forced Prosper to compute $[P(\alpha)]$, $[H(\alpha)]$, ... as polys of right degree.

Remains to check $P(\alpha+2)-P(\alpha+1)-P(\alpha) = D(\alpha)\cdot H(\alpha)$, using the encodings.

No problem. this is a quadratic equation. Check:

$$e([P(\alpha+2)-P(\alpha+1)-P(\alpha)],[1]) = e([D(\alpha)],[H(\alpha)])$$

Conclusion. Since $P(\alpha)$, $H(\alpha)$ etc are polys of right degree, original argument applies: checking equality at random α ensures with $\geq 1\text{-}1000/|\mathbb{Z}_p| > 99\%$ probability the equality is true on the whole polys → D divides $P(\alpha+2)-P(\alpha+1)-P(\alpha)$ → computation was correct.

Efficiency

Prosper proves correct computation by providing a **constant number** of encodings: $[P(\alpha)]$, $[\gamma P(\alpha)]$, $[H(\alpha)]$, $[\eta H(\alpha)]$ etc.

#encodings is absolute constant, independent of circuit size.

Pre-processing by Véronique was still linear in circuit size: publishes $[\alpha^i]$, $i \leq 1000$, etc. But...

- Can be amortized over many circuits.
- Exist “fully succint” SNARKs, with O(log(circ. size)) verifier pre-processing.

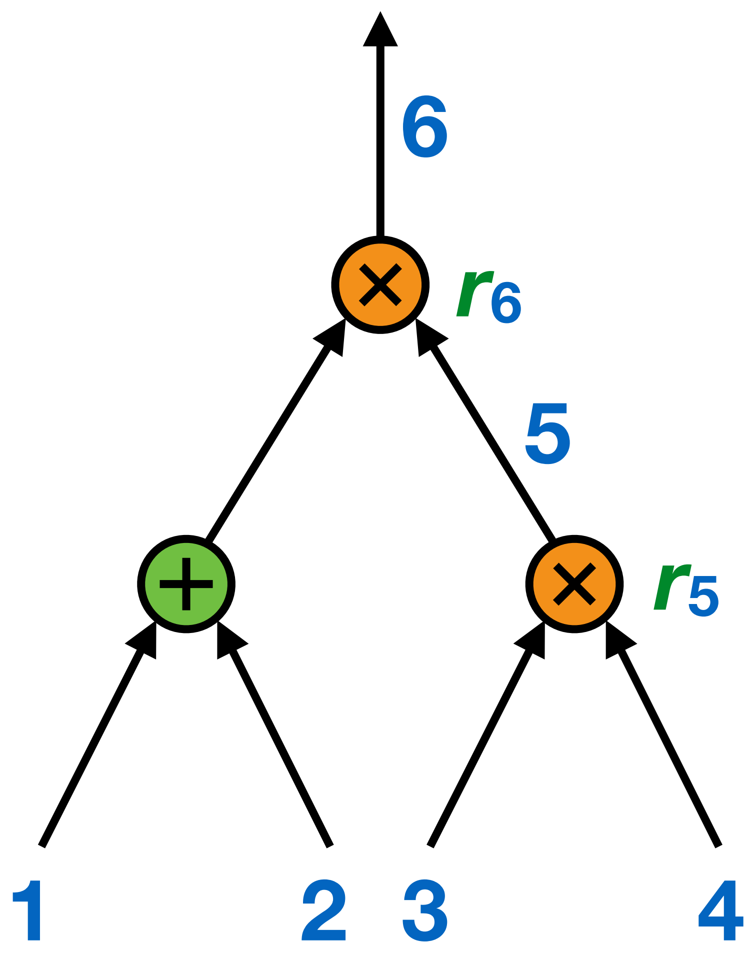
Working with circuits directly

In essence: we have seen how to do a succint proof of polynomial divisibility.

Can in principle encode valid machine state transitions as polynomial constraints → succint proofs for circuit-SAT.

Now: want to do that more concretely = get SNARKs for circuit-SAT (directly).

We are going to encode a circuit as polynomials.



For simplicity, forget about negations. Write circuit with ⊕ (XOR), ⊗ (AND) gates. Then:

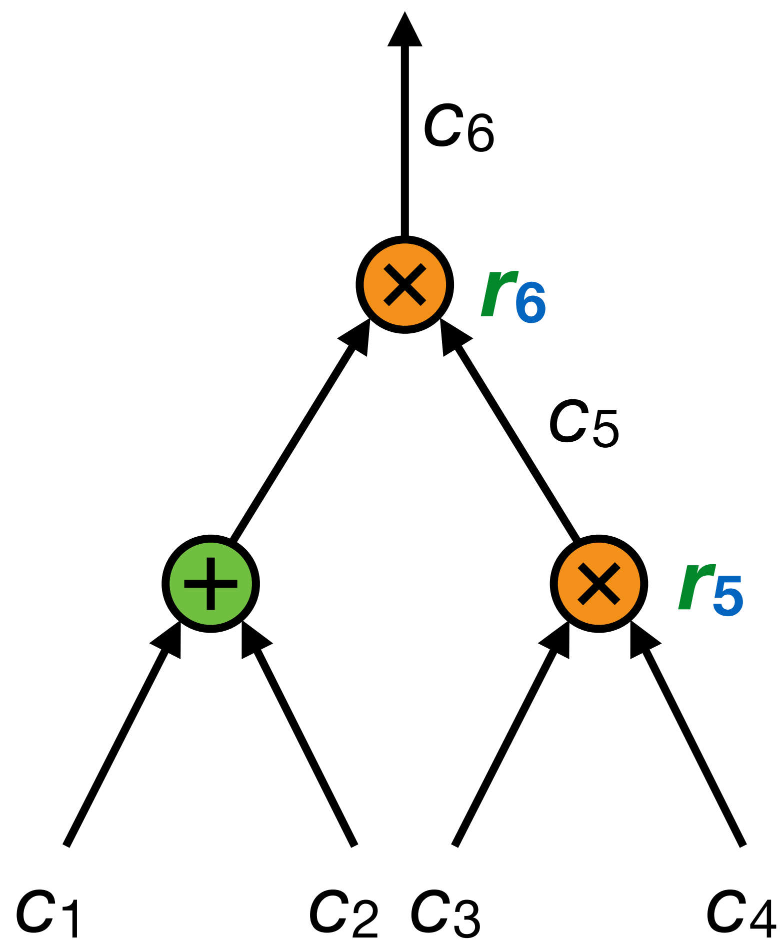
1) Associate an integer **i** to each input; and to each output of a mult gate ⊗.

2) Associate an element $r_i \in \mathbb{F}_q$ to mult gate **i**.

Now circuit can be encoded as polys. For each **i** = **1**,...,**6**, define polynomials $\mathbf{v}_i$, $\mathbf{w}_i$, $\mathbf{y}_i$:

- $\mathbf{v}_i(r_j)$=1 if value **i** is *left input* to gate **j**, 0 if not.
- $\mathbf{w}_i(r_j)$=1 if value **i** is *right input* to gate **j**, 0 if not.
- $\mathbf{y}_i(r_j)$=1 if value **i** is *output* of gate **j**, 0 if not.

Exemple.



In this case, $\mathbf{v}_i$, $\mathbf{w}_i$, $\mathbf{y}_i$ are degree 2.

Encoding mult gate **5**:

- $\mathbf{v}_3(r_5)=1$, $\mathbf{v}_i(r_5)=0$ otherwise.
- $\mathbf{w}_4(r_5)=1$, $\mathbf{w}_i(r_5)=0$ otherwise.
- $\mathbf{y}_5(r_5)=1$, $\mathbf{y}_i(r_5)=0$ otherwise.

Encoding mult gate **6**:

- $\mathbf{v}_1(r_6)=\mathbf{v}_2(r_6)=1$, $\mathbf{v}_i(r_6)=0$ otherwise.
- $\mathbf{w}_5(r_6)=1$, $\mathbf{w}_i(r_6)=0$ otherwise.
- $\mathbf{y}_6(r_6)=1$, $\mathbf{y}_i(r_6)=0$ otherwise.

The point: an assignment of variables c_1, ..., c_6 satisfies the circuit iff:

$$(\Sigma c_i \mathbf{v}_i(r_5)) \cdot (\Sigma c_i \mathbf{w}_i(r_5)) = \Sigma c_i \mathbf{y}_i(r_5) \quad \text{and} \quad (\Sigma c_i \mathbf{v}_i(r_6)) \cdot (\Sigma c_i \mathbf{w}_i(r_6)) = \Sigma c_i \mathbf{y}_i(r_6)$$

Equivalently:

$(X-r_5)(X-r_6)$ divides $(\Sigma c_i \mathbf{v}_i) \cdot (\Sigma c_i \mathbf{w}_i) - \Sigma c_i \mathbf{y}_i$

→ we have reduced:

"Prosper wants to prove he knows inputs satisfying a circuit."

into:

"Prosper wants to prove he knows linear combinations $V = \Sigma c_i \mathbf{v}_i$, $W = \Sigma c_i \mathbf{w}_i$, $Y = \Sigma c_i \mathbf{y}_i$, *such that* $T = (X-\mathbf{r}_5)(X-\mathbf{r}_6)$ *divides* $VW-Y$*."*

$$\Leftrightarrow \exists\, H,\ \ T \cdot H = V \cdot W - Y$$

1. quadratic!
2. polynomial equality!

We know how to do that!

V. publishes $[\alpha^i]$, plus auxiliary $[\gamma\alpha^i]$ etc... (at setup, indep. of circuit)

P.'s proof is $[V(\alpha)]$, $[W(\alpha)]$, $[Y(\alpha)]$, $[H(\alpha)]$, plus auxiliary $[\gamma V(\alpha)]$ etc...

V. checks $e(T(\alpha),H(\alpha))=e([V(\alpha)],[W(\alpha)])e([Y(\alpha)],[1])^{-1}$ and auxiliary stuff.

Constant-size proof. Construction works for any circuit.

In practice

Construction was proposed in Pinocchio scheme (Parno et al. S&P 2013).

Practical: proofs ~ 300kB, verification time ~ 10 ms.

- Introduced for verifiable outsourced computation.
- Further improvements since.

Can be made zero-knowledge at negligible additional cost.

A ZK application: e-Voting

e-Voting

Are going to see (more or less) Helios voting system.

https://heliosvoting.org/

Used for many small- to medium-scale elections.
Including IACR (International Association for Cryptologic Research).

We will focus on yes/no referendum.

Nice description of Belenios variant: https://hal.inria.fr/hal-02066930/document

Goals

We want:

- Vote privacy
- Full verifiability:
  - Voter can check their vote was counted
  - Everyone can check election result is correct

    Every voter cast ≤1 vote, result = number of yes votes

We do not try to protect against:

- Coercion/vote buying

Nice description of Belenios variant: https://hal.inria.fr/hal-02066930/document

Basics

Election = want to add up encrypted votes...

→ just use additively homomorphic encryption!

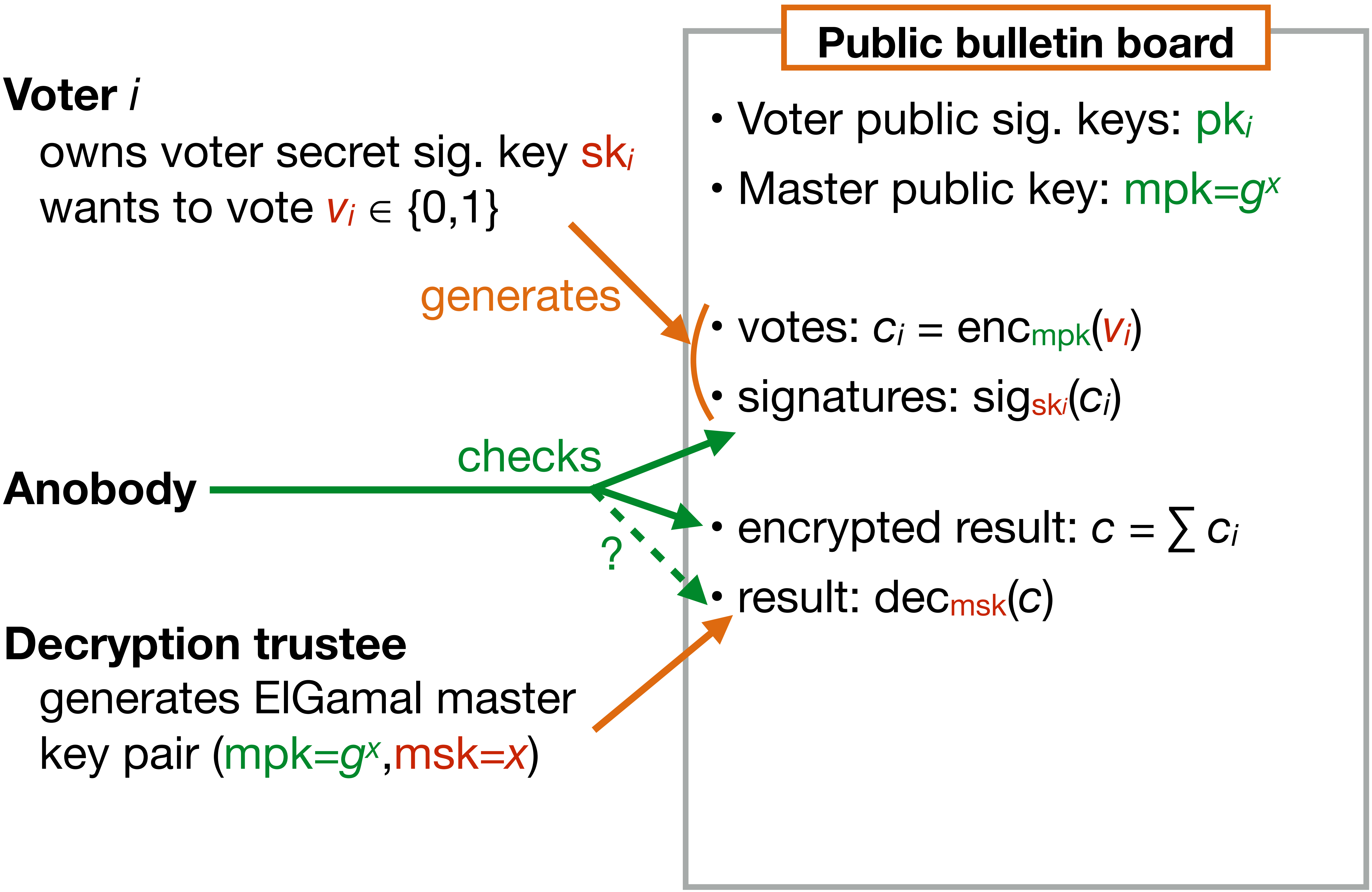
Helios: use ElGamal. Multiplicatively homomorphic.
To make it additive: vote for v is g^v.
Recovering v from g^v is discrete log, but brute force OK (v small).

In addition: voters sign their votes.

Helios: Schnorr signatures.

Who decrypts the result?

First attempt



Problem: how to verify final result.

Making election result verifiable

ElGamal encryption:

Master keys: (mpk=g^x,msk=x)

Encrypted election result $c = (c_L = g^k, c_R = m \cdot g^{xk})$

Election result = Dec(c) = $m = c_R \,/\, c_L{}^x$

→ giving decryption is same as giving $c_L{}^x$

→ to prove decryption is correct, prove:

discrete log of $(c_L)^x$ in base c_L = discrete log of mpk=g^x in base g

⇔ $(g, g^x, c_L, c_L{}^x) \in$ Diffie-Hellman language

→ to make election result verifiable: decryption trustee just provides NIZK proof of DH language for $(g, g^x, c_L, c_L{}^x)$!

Take ZK proof of DH language from earlier + Fiat-Shamir → NIZK

Note ZK property is crucial.

Now with verifiable election result

Voter i
owns voter secret sig. key sk_i
wants to vote $v_i \in \{0,1\}$

generates

Anobody

checks

Decryption trustee
generates ElGamal master key pair ($mpk=g^x$, $msk=x$)

Public bulletin board

- Voter public sig. keys: pk_i
- Master public key: $mpk=g^x$
- votes: $c_i = enc_{mpk}(v_i)$
- signatures: $sig_{sk_i}(c_i)$
- encrypted result: $c = \sum c_i$
- result: $dec_{msk}(c)$ + DH proof

Problem 2: how about I vote $enc_{mpk}(1000)$?

Proving individual vote correctness

In addition to vote $\text{enc}_{\text{mpk}}(v_i)$ and signature $\text{sig}_{\text{sk}_i}(c_i)$, voter provides NIZK proof that $v_i \in \{0,1\}$.

Helios doesn't use SNARK here, but more tailored proof of disjunction.

Note ZK property is crucial again.

To prevent “weeding attack” (vote replication):

NIZK proof includes g^k, pk_i in challenge randomness (hash input of sigma protocol), where g^k is the randomness used in $\text{enc}_{\text{mpk}}(v_i)$.

→ proof (hence vote) cannot be duplicated without knowing sk_i.

Now with full verifiability

Voter i
owns voter secret sig. key sk_i
wants to vote $v_i \in \{0,1\}$

generates

Anobody

checks

Decryption trustee
generates ElGamal master key pair ($mpk=g^x$, $msk=x$)

Public bulletin board

- Voter public sig. keys: pk_i
- Master public key: $mpk=g^x$
- votes: $c_i = enc_{mpk}(v_i)$ + proof ≤ 1
- signatures: $sig_{sk_i}(c_i)$
- encrypted result: $c = \sum c_i$
- result: $dec_{msk}(c)$ + DH proof

Bonus problem: replace decryption trustee by threshold scheme.