

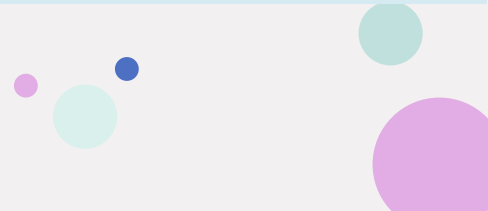


Lecture 20

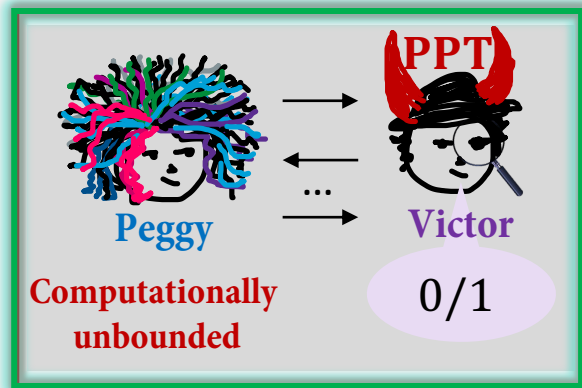
Proofs of Knowledge, Non-interactive ZK, Sigma Protocols

CS409: Introduction to Cryptography

Sruthi Sekar



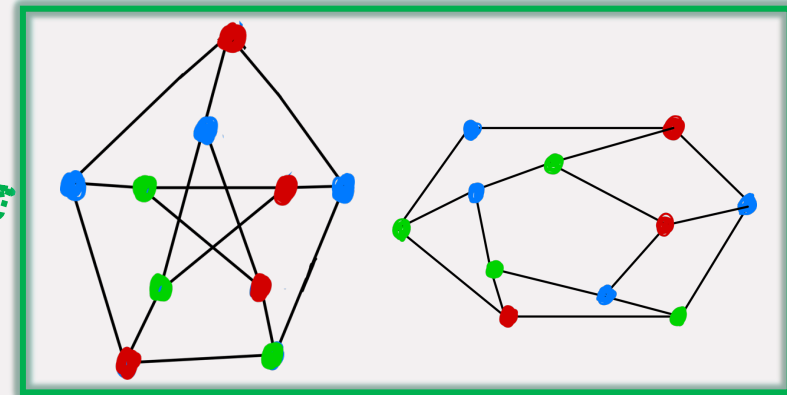
Recap: Last Class



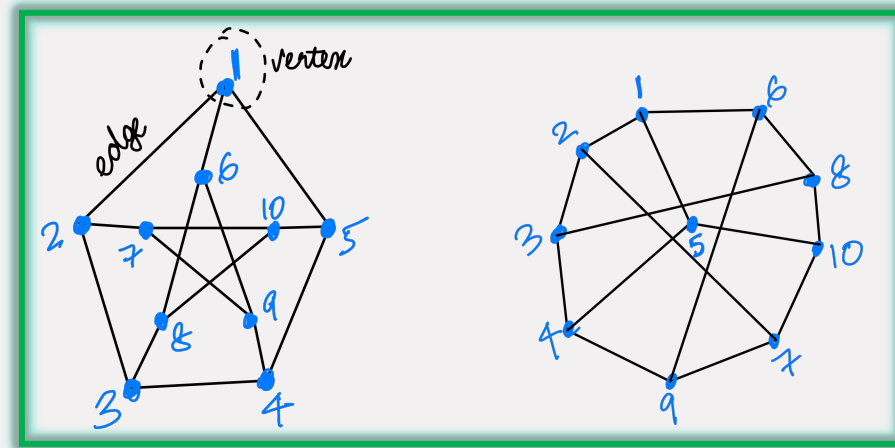
Malicious Verifier ZK Proof

For all $L \in \text{NP}$

Using
**Commitment
Schemes**

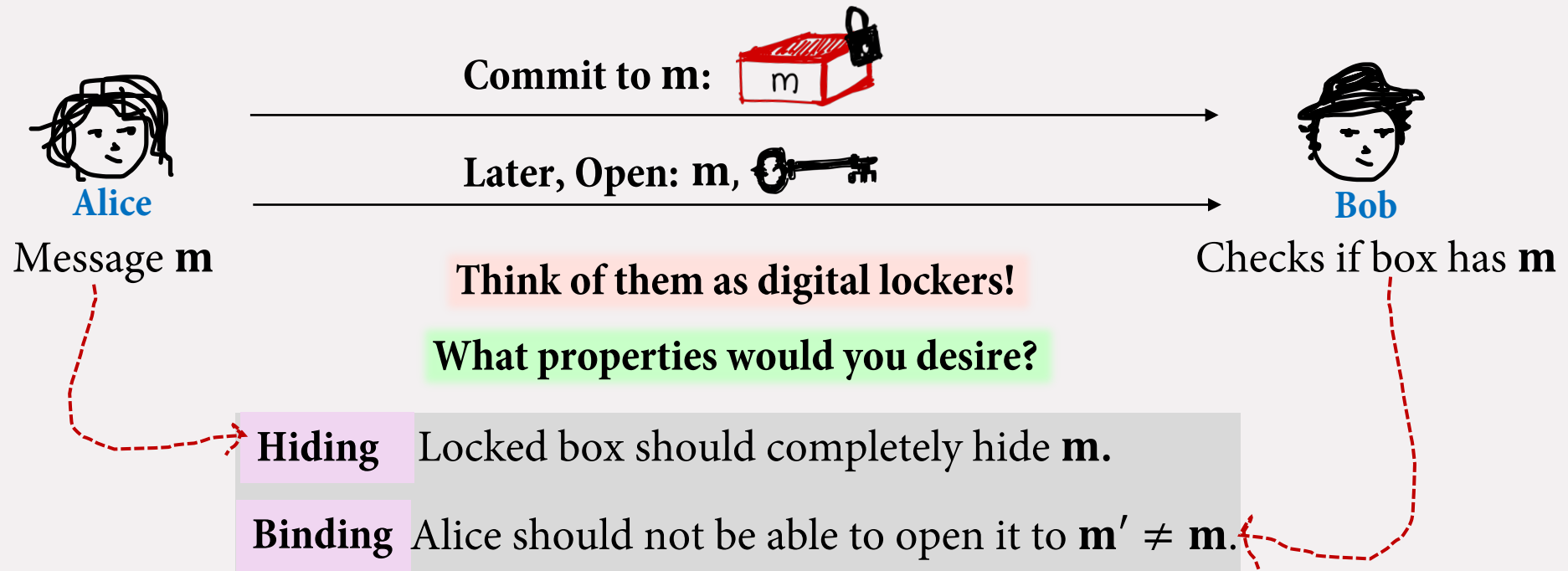


Graph 3-Coloring



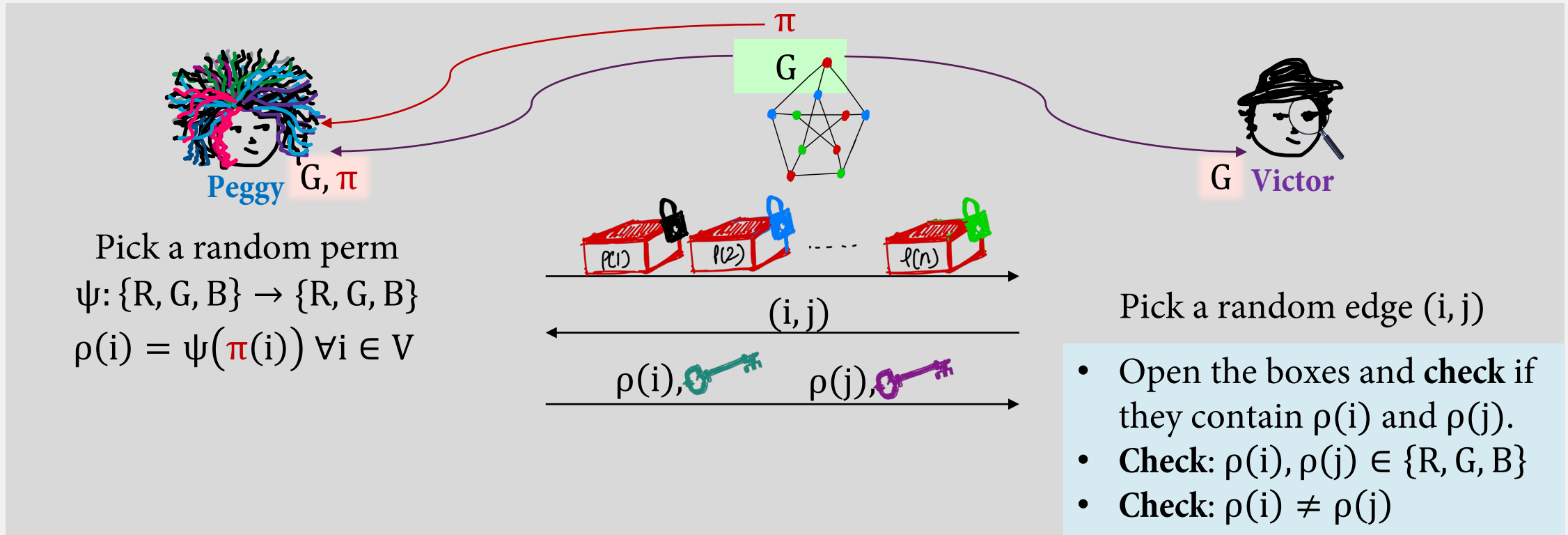
Graph Isomorphism (GI)

Recall: Commitment Schemes



We will formalize this later but say we had this digital locker, let's build a ZKP for G3C

ZKP for G3C



ZKP for Graph 3-Coloring (G3C): Π_{G3C}

$L_{G3C} = \{G: \exists \text{ coloring assignment } \pi: V \rightarrow \{R, G, B\} \text{ to } G \text{ such that neighbors have distinct colors}\}$

****** $V = \{1, 2, \dots, n\}$

ZKP for G3C

Theorem 1

If the *digital lockers* have *perfect binding and computational hiding*, then Π_{G3C} is a **malicious-verifier computational ZKP** for L_{G3C} .

Proof Sketch

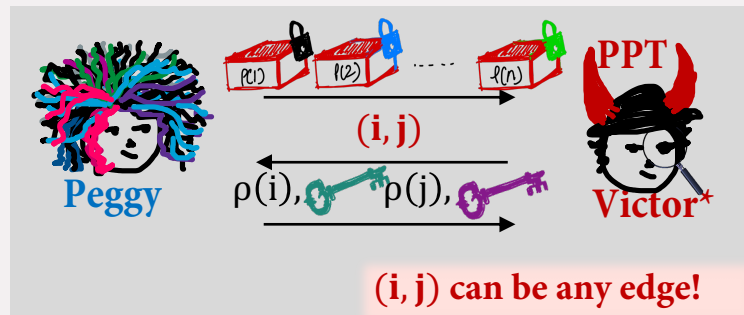
Completeness H.W.

Soundness G not 3-colorable $\Rightarrow \exists (i, j)$ (an edge) such that $\pi(i)$ and $\pi(j)$ are the same color!

Binding? V will catch this edge and reject with probability $\geq 1/|E|$ (E is set of edges).

(*Sequentially*)^{*} repeat $|E| \cdot t$ times so that V accepts with probability $\leq (1 - 1/|E|)^{|E| \cdot t} \leq 2^{-t}$

Malicious Verifier Comp ZK



****For one run (not the sequential repeat setting)**

1. Pick random (i^*, j^*) . Color these with different colors and the rest of the vertices red (ρ be this map).
2. Send $\rho(1)$, $\rho(2)$, ..., $\rho(n)$ to V^* and get the edge (i, j) .
3. If $(i^*, j^*) = (i, j)$ life is good, **output**: $\rho(i)$, $\rho(j)$.
4. Else, repeat!

Hiding? ****Use hiding to show that the simulated transcript is indistinguishable from the real transcript (only for single run!)**

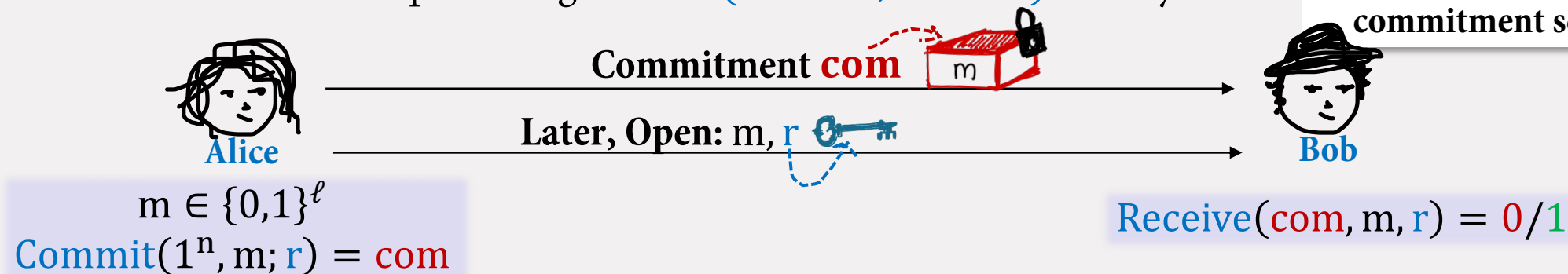
Commitment Schemes

HW 1

Can you get both perfect hiding and perfect binding simultaneously from a commitment scheme?

Definition 1: Commitment Scheme

A commitment scheme is a pair of algorithms (**Commit**, **Receive**) with syntax:



Correctness: For all $n \in \mathbb{N}$ and inputs $m \in \{0,1\}^\ell$: $\Pr_r[\text{Receive}(\text{Commit}(1^n, m; r), m, r) = 1] = 1$

Computational Hiding: For any $m_0 \neq m_1 \in \{0,1\}^\ell$: $\text{Commit}(1^n, m_0; r_0) \approx_c \text{Commit}(1^n, m_1; r_1)$

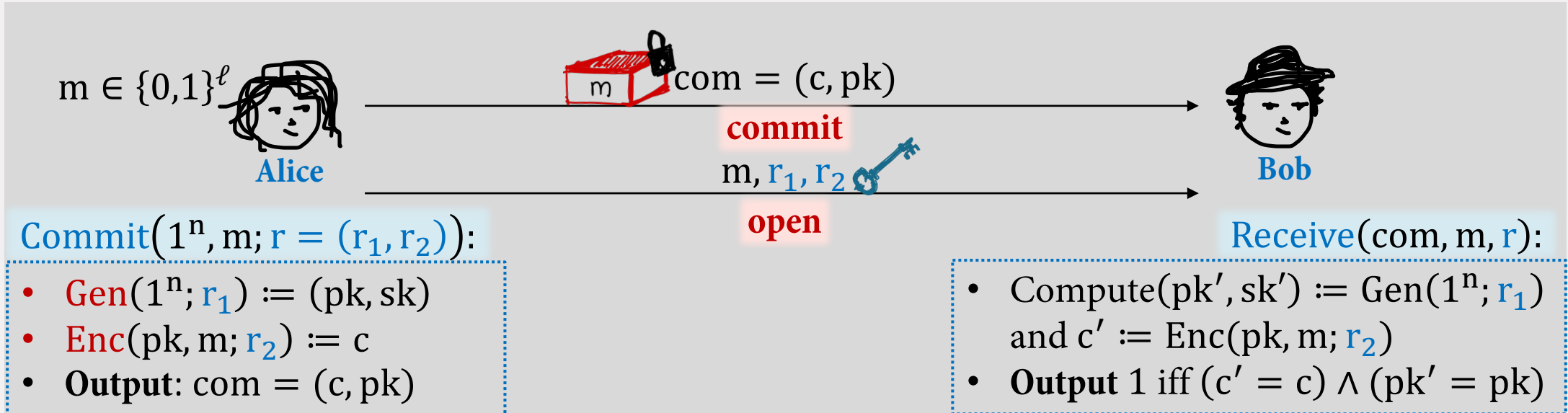
Perfect Binding: For any $\text{com} \in \{0,1\}^*$, there **do not exist** two openings (keys) r_1 and r_2 such that $\text{Receive}(\text{com}, m_1, r_1) = 1 = \text{Receive}(\text{com}, m_2, r_2)$ for distinct m_1 and m_2

****Remark:** The commit phase can be interactive as well!

****Commit**($1^n, m; r$): denotes the algorithm **Commit** taking input $(1^n, m)$ as input, using randomness **r** to output **com**

Commitment Schemes: Construction 1

PKE $\Pi = (\text{Gen}, \text{Enc}, \text{Dec}) \Rightarrow$ Commitment Scheme $\Sigma = (\text{Commit}, \text{Reveal})$



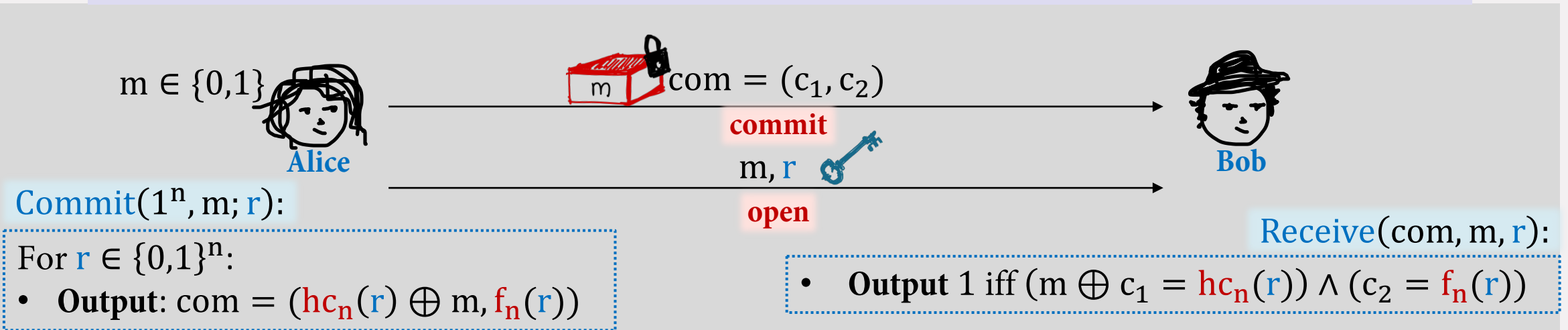
Properties from Π ?

- Should be able to recognize honestly sampled pk .
- IND-CPA security $\Rightarrow$ Computational hiding
- Perfect Correctness of decryption $\Rightarrow$ Perfect binding

(Proof discuss)

Commitment Schemes: Construction 2

OWP family $\{f_n: \{0, 1\}^n \rightarrow \{0, 1\}^n\}$ with hard-core predicate $hc \Rightarrow$ Bit-commitment Scheme Σ

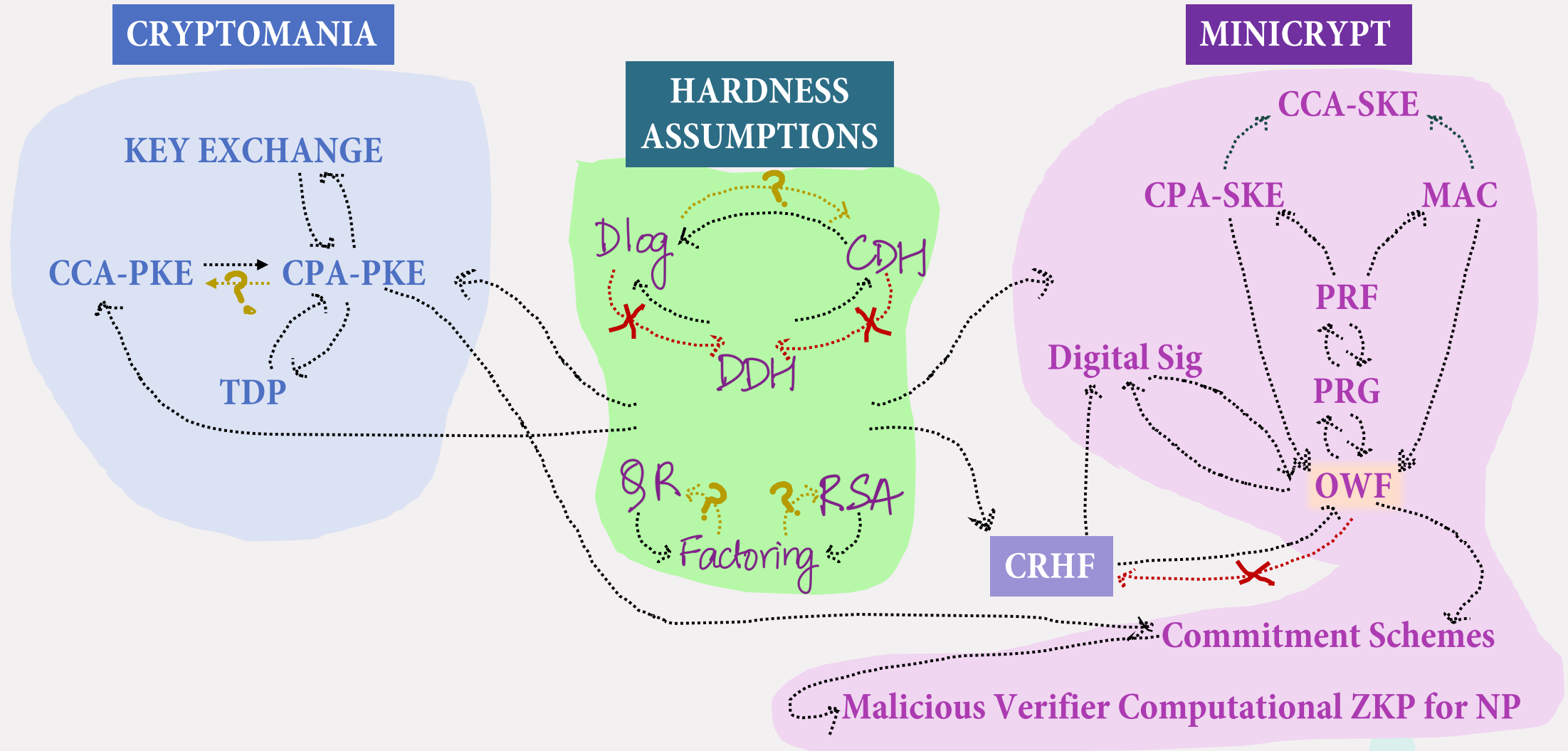


- Security of hard-core predicate $hc \Rightarrow$ Computational hiding [\[Lec15, Construction 1\]](#)
- f_n permutation $\Rightarrow$ Perfect binding (Why?)

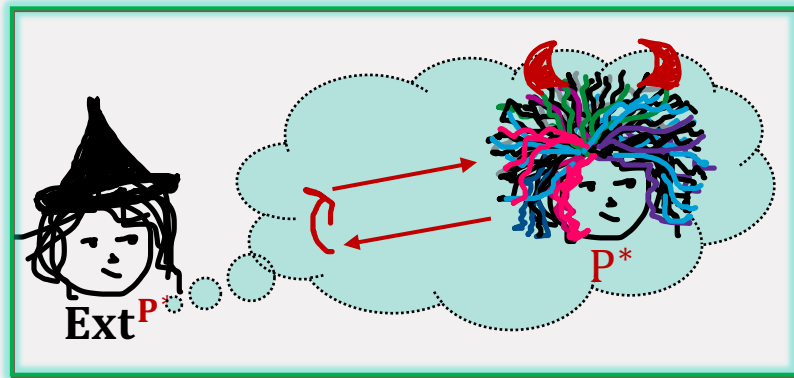
HW 2

1. Formalize the proof!
2. How to extend this bit-commitment to string-commitment?

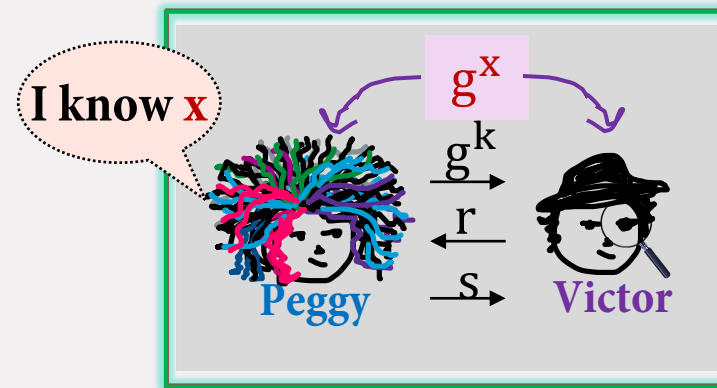
Big Picture so far



Today

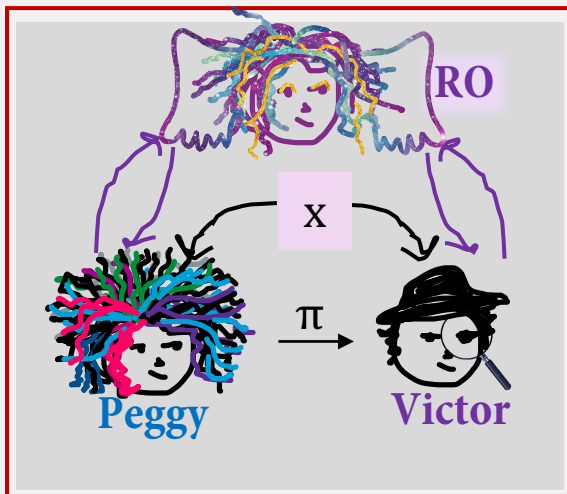


ZK Proof of Knowledge

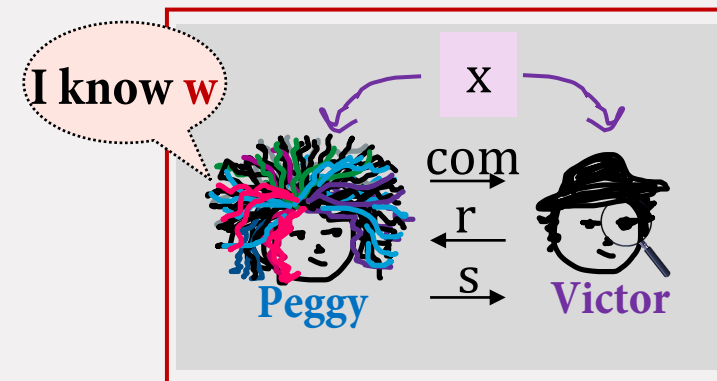


Schnorr's ZKPoK for Dlog

Generalized to

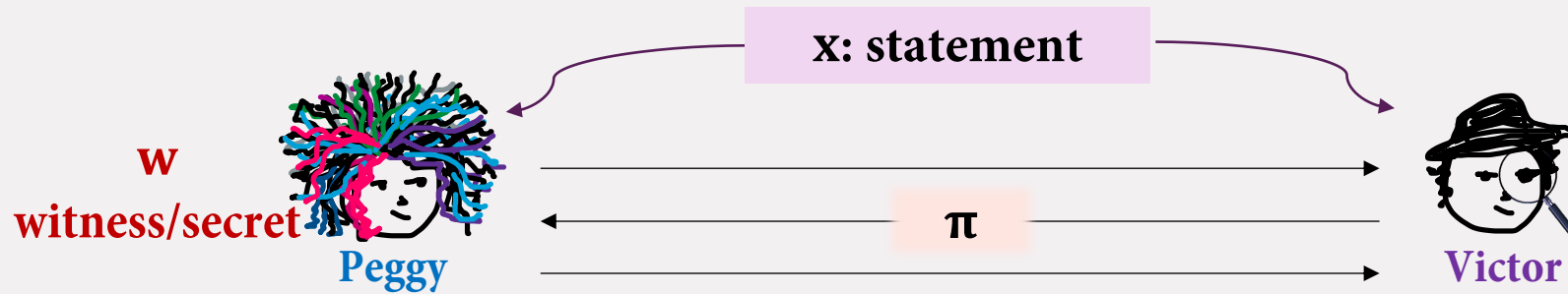


Non-interactive ZK (NIZK)



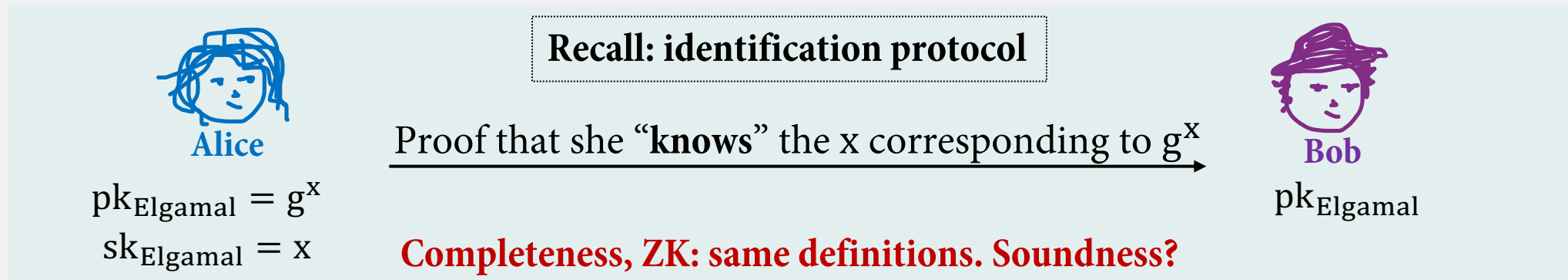
Sigma Protocols

ZK Proof of Knowledge



ZK: Peggy proves that $x \in L$ without revealing w

Sometimes you need a stronger property from π : it should prove that Peggy **“knows”** a witness!



How do we quantify “knowledge”? It should be possible to “extract a witness” from Alice!

Defining ZK Proof of Knowledge

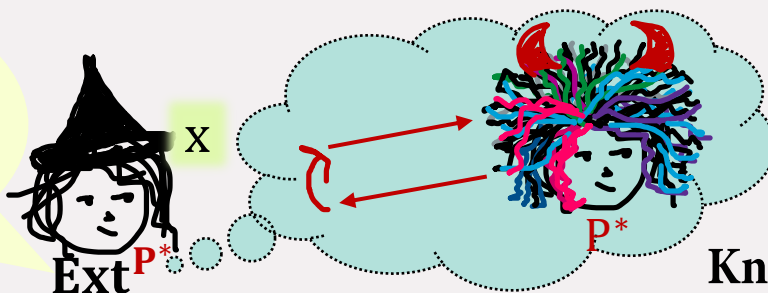
Definition 2: ZK Proof of Knowledge (zkPoK)

An interactive protocol $\Pi = (P, V)$ for an NP language L is a **zero-knowledge proof of knowledge** if it is

1. Complete
2. Zero-knowledge
3. **Knowledge Sound**: $\exists$ expected polynomial-time extractor Ext such that $\forall$ prover P^* and instance x :

$$\Pr_{w \leftarrow \text{Ext}^{P^*}(x)} [w \text{ is a witness for } x] \geq \Pr[\langle P^*, V \rangle(x) \rightarrow 1] - \epsilon_k \rightarrow \text{Knowledge error}$$

If P^* convinced V to accept then she will get w at the end of this!



Knowledge soundness $\implies$ Soundness Why? H.W.

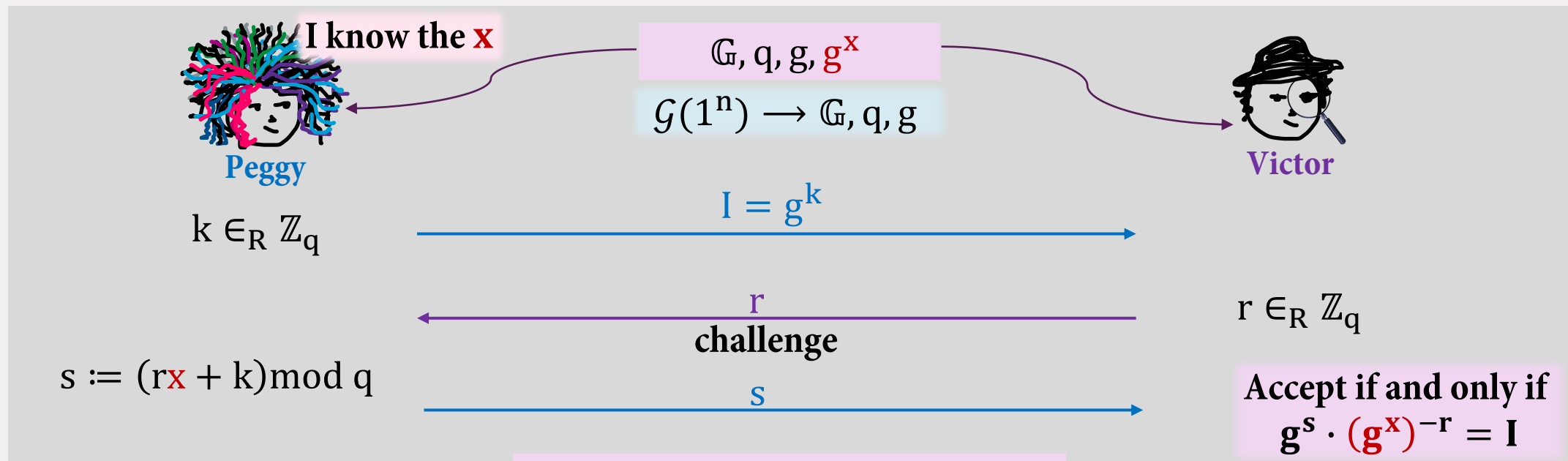
- If you ask for one out of 2 and 3, we can trivially get a scheme. **Why?**
- Ext has some extra power over V , or else one can break ZK using Ext ! E.g., **Power of rewinding P^***

ZK PoK for Dlog: Schnorr's Protocol

Discrete Log Assumption (RECALL)

The **discrete log problem is hard relative to $\mathcal{G}$** if for all PPT Logan, there exists a negligible negl such that

$$\Pr_{\substack{\mathcal{G}(1^n) \rightarrow \mathbb{G}, q, g \\ h \in_R \mathbb{G}}} [\text{Logan}(\mathbb{G}, q, g, h) \rightarrow x \text{ such that } x \in \mathbb{Z}_q \wedge g^x = h] \leq \text{negl}(n)$$



Schnorr's ZKPoK for L_{Dlog} : Π_{Dlog}

$$L_{\text{Dlog}} = \{h \in \mathbb{G} : \exists x \in \mathbb{Z}_q \text{ s.t. } g^x = h\}$$

ZK PoK for Dlog: Schnorr's Protocol

Theorem 2: Schnorr ZKPoK

Π_{Dlog} is a ZKPoK for L_{DLog} with $\epsilon_k \leq 1/q$

Proof Sketch:

Completeness: $g^s \cdot (g^x)^{-r} = g^{(s-xr)} = g^k = I \ (\because s - xr = k \bmod q)$

Honest Verifier Zero-Knowledge (recall):

$\text{View}_V(\langle P, V \rangle(g^x)) := (g^k, r, s = rx + k \bmod q) \equiv$



$\text{Sim}(g^x)$: (out-of-order) pick $r, s \in_R \mathbb{Z}_q$
 set $I = g^s \cdot (g^x)^{-r}$
 output: (I, r, s)

Proof of Knowledge (recall):

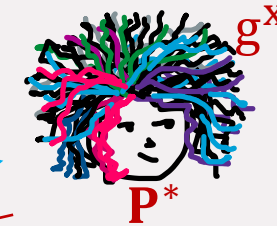
$\text{Ext}^{P^*}(g^x)$: Invoke P^* on g^x (Goal: extract x)

- P^* sends I .
- Send challenge r_1 and get s_1 .
- Rewind** P^* send r_2 and get s_2 .

Output: $(s_1 - s_2)(r_1 - r_2)^{-1} \bmod q$



Ext^{P^*}

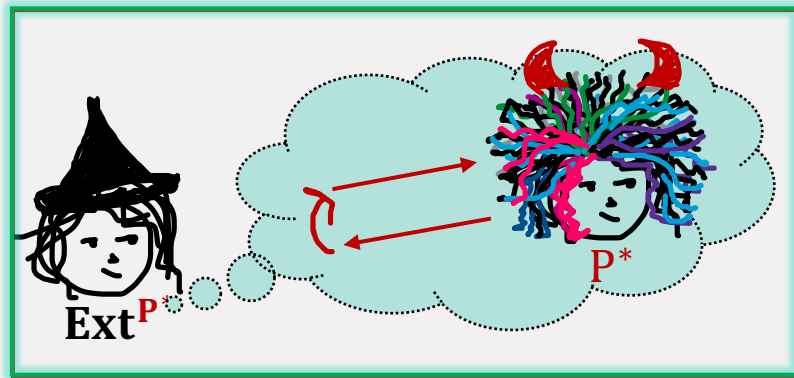


Recall from [Thm 3, Lec 17]

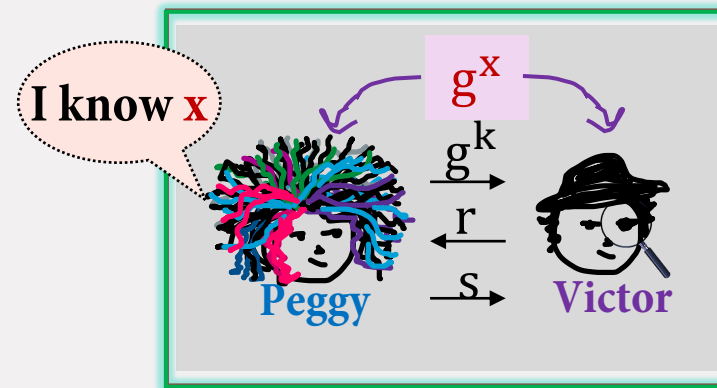
$\epsilon_k \leq 1/q$

If P^* gives accepting responses $\wedge (r_1 \neq r_2)$ then $g^{s_1 - xr_1} = I = g^{s_2 - xr_2} \Rightarrow x = (s_1 - s_2)(r_1 - r_2)^{-1} \bmod q$

Today

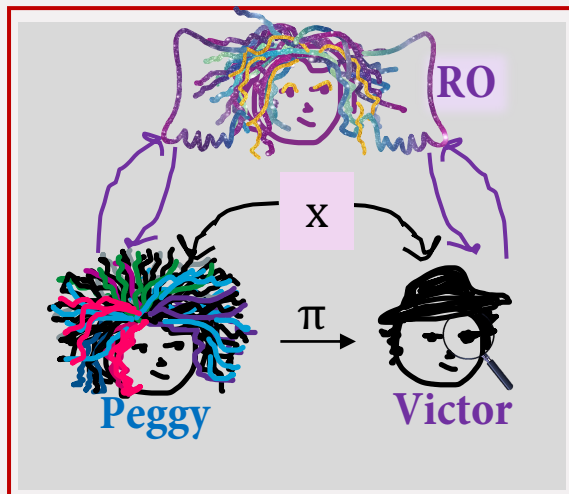


ZK Proof of Knowledge

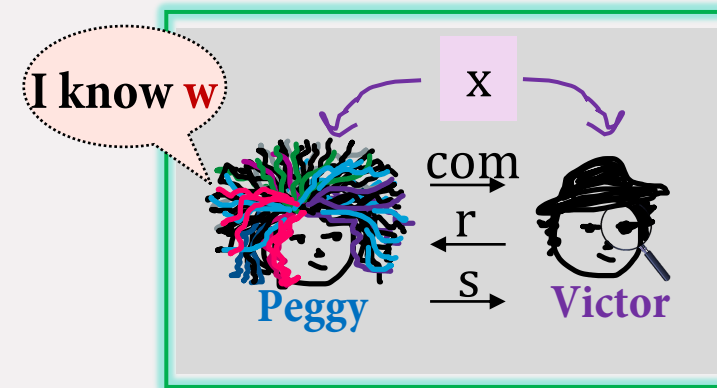


Schnorr's ZKPoK for Dlog

Generalized to

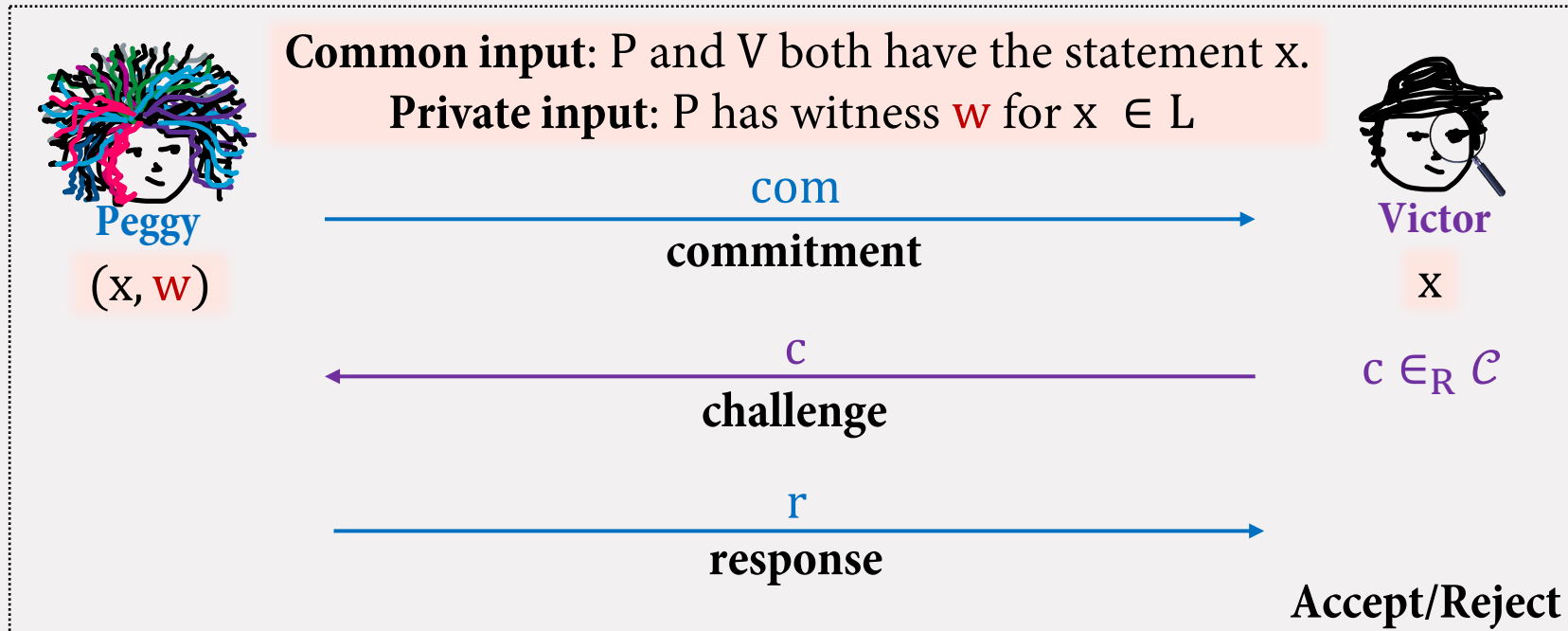


Non-interactive ZK (NIZK)



Sigma Protocols

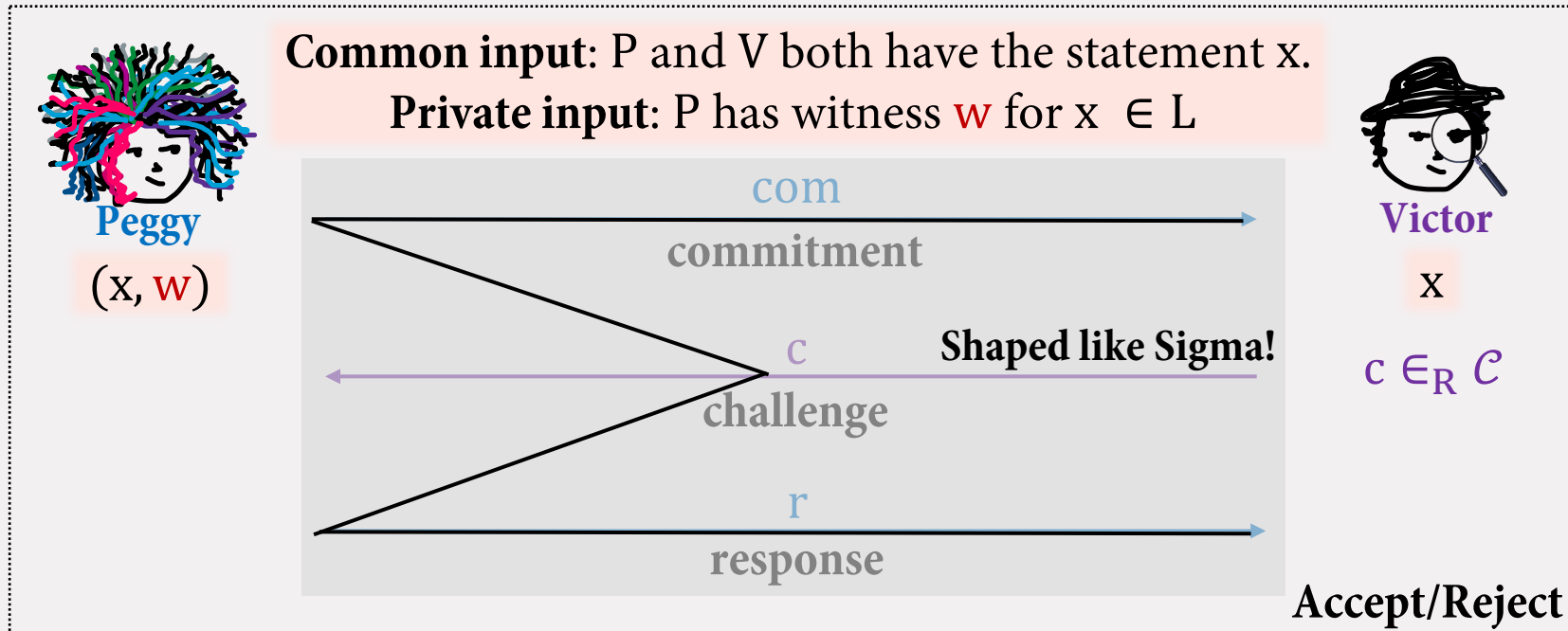
Sigma Protocols



Three-round protocol:

- P sends a commitment com to V.
- V sends a random challenge from some finite challenge space $\mathcal{C}$.
- P then sends a response r to the challenge.
- V either accepts or rejects based on x and the conversation (com, c, r) .

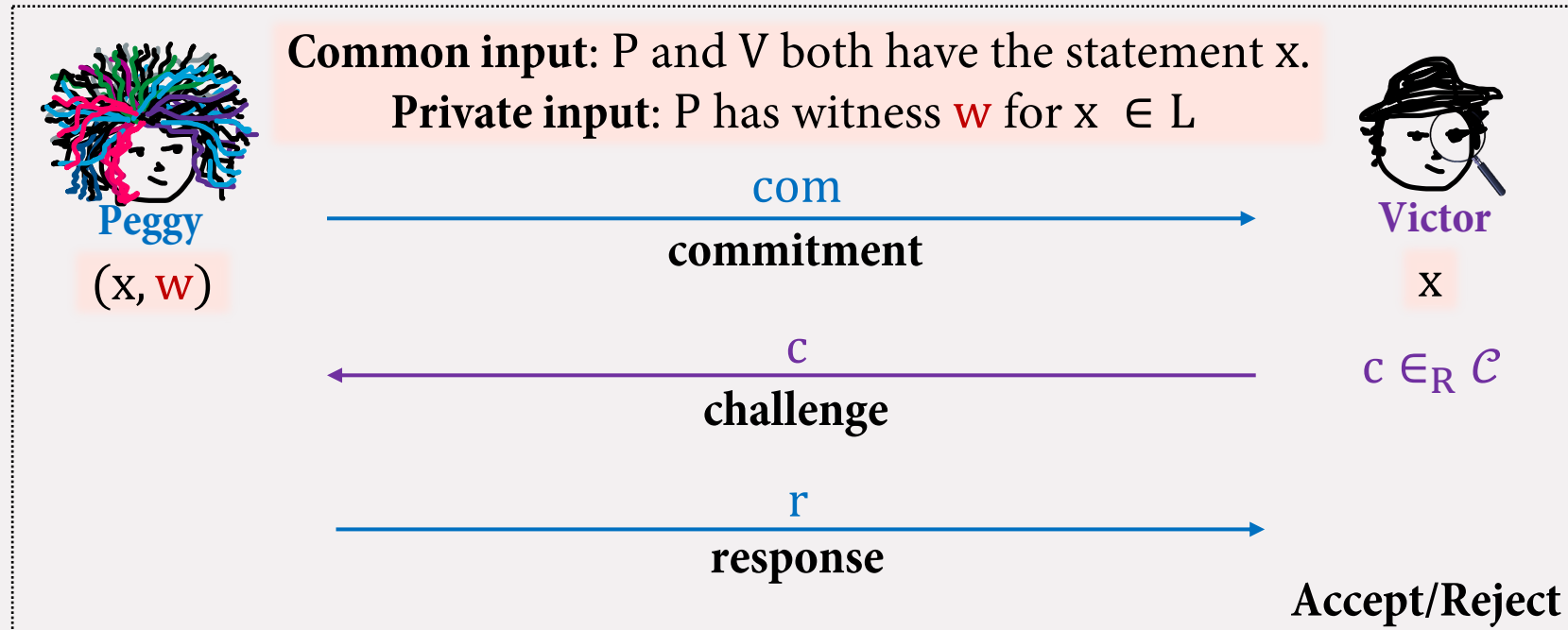
Sigma Protocols



Three-round protocol:

- P sends a commitment **com** to V.
- V sends a random challenge from some finite challenge space $\mathcal{C}$.
- P then sends a response **r** to the challenge.
- V either accepts or rejects based on x and the **conversation** (**com**, **c**, **r**).

Sigma Protocols: Definitions



Completeness:

$\forall (x, w) \in \mathcal{R}_L, \langle P(x, w), V(x) \rangle$ accepts with probability 1

Special Soundness:

$\exists$ and efficient **witness extractor Ext** such that for $c \neq c'$ and $(com, c, r), (com, c', r')$ both accepting
 $Ext(x, (com, c, r), (com, c', r')) = w$ such that $(x, w) \in \mathcal{R}_L$

HVZK: Same as before [Def 3, Lec 18]!

Schnorr is a Σ -protocol

**** $\mathcal{R}_L$: relation corresponding to the language L**

Sigma Protocol for DH Tuple: Chaum-Pedersen

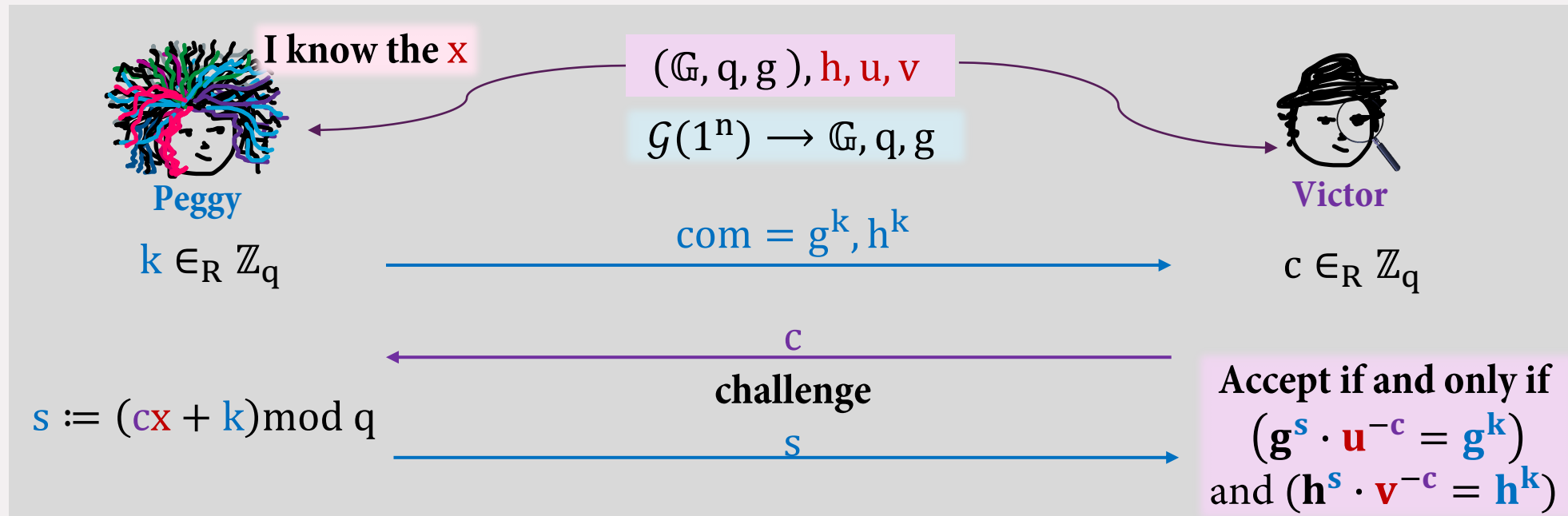
Diffie Hellman Triple

For $\mathcal{G}(1^n) \rightarrow \mathbb{G}, q, g, (g^x, g^y, g^z)$ is a **DH triple** if $z = xy$

$$\mathcal{R}_{\text{DH}} = \{((h, u, v), \mathbf{x}) \in \mathbb{G}^3 \times \mathbb{Z}_q : u = g^{\mathbf{x}} \text{ and } v = h^{\mathbf{x}}\}$$

statement

witness



Chaum-Pedersen Σ protocol for L_{DH} : Π_{DH}

$$L_{\text{DH}} := \{(h, u, v) \in \mathbb{G}^3 : \exists x \in \mathbb{Z}_q \text{ s. t. } ((h, u, v), x) \in \mathcal{R}_{\text{DH}}\}$$

Sigma Protocol for DH Tuple: Chaum-Pedersen

Theorem 3: Chaum-Pedersen Σ protocol

Π_{DH} is a Σ -protocol for $\mathcal{R}_{\text{DH}}$.

Proof Sketch

Completeness: H.W.

Honest Verifier Zero-Knowledge:

$\text{View}_V(\langle P, V \rangle(\mathbf{h}, \mathbf{u}, \mathbf{v})) := ((g^k, h^k), \mathbf{c}, \mathbf{s} = \mathbf{c}\mathbf{x} + \mathbf{k} \bmod q)$



$\text{Sim}(\mathbf{h}, \mathbf{u}, \mathbf{v})$: (out-of-order) pick $c, s \in_R \mathbb{Z}_q$
output: $((g^s \cdot \mathbf{u}^{-c}), \mathbf{h}^s \cdot \mathbf{v}^{-c}), c, s)$

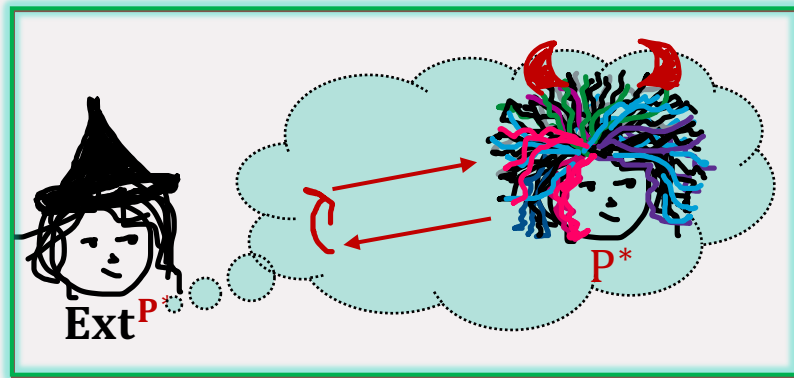
Special Soundness: (exactly like Schnorr)



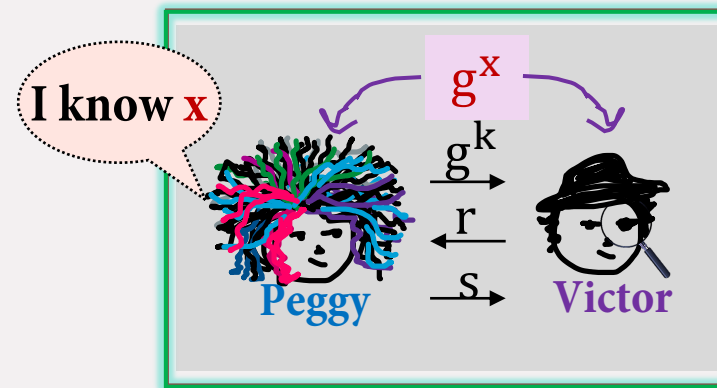
$\text{Ext}((\mathbf{h}, \mathbf{u}, \mathbf{v}), (\text{com} = (g^k, h^k), \mathbf{c}, \mathbf{s}), (\text{com}, \mathbf{c}', \mathbf{s}'))$:

- We know $\mathbf{c} \neq \mathbf{c}'$ and both are accepting transcripts.
 $\because g^{\mathbf{s}} \mathbf{u}^{-\mathbf{c}} = g^{\mathbf{k}} = g^{\mathbf{s}'} \mathbf{u}^{-\mathbf{c}'} \text{ \& } \mathbf{h}^{\mathbf{s}} \mathbf{v}^{-\mathbf{c}} = \mathbf{h}^{\mathbf{k}} = \mathbf{h}^{\mathbf{s}'} \mathbf{v}^{-\mathbf{c}'}$
- $\Rightarrow \log_g \mathbf{u} = \log_h \mathbf{v} = (\mathbf{s} - \mathbf{s}')(\mathbf{c} - \mathbf{c}')^{-1} \bmod q$

Today

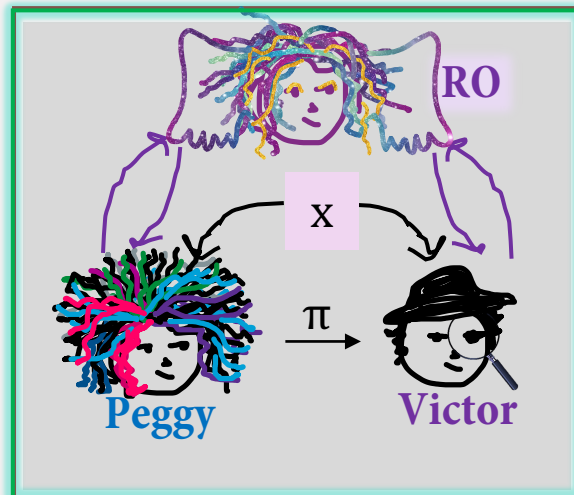


ZK Proof of Knowledge

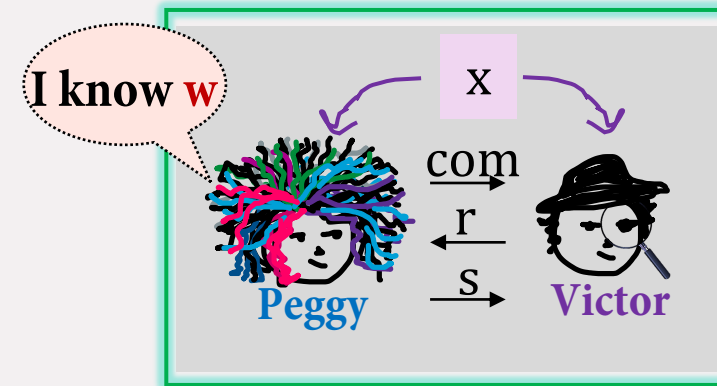


Schnorr's ZKPoK for Dlog

Generalized to

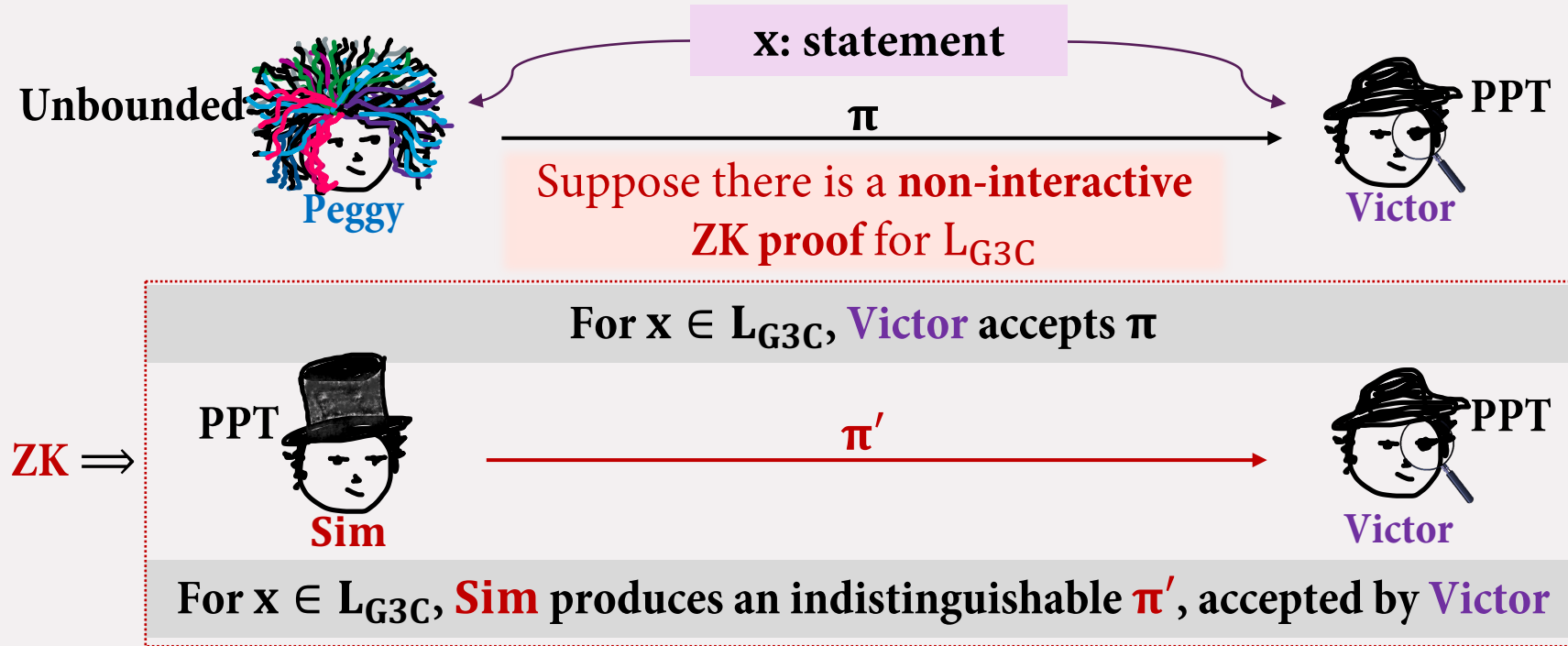


Non-interactive ZK (NIZK)



Sigma Protocols

Is Interaction Necessary for ZK?



What if you ran **Sim** over $x \notin L_{G3C}$?

For $x \notin L_{G3C}$, **Sim** still produces π' , accepted by Victor!

Why?? If **Sim** (PPT) produces an invalid π' then it can be used to tell if $x \in L_{G3C}$ or not!*

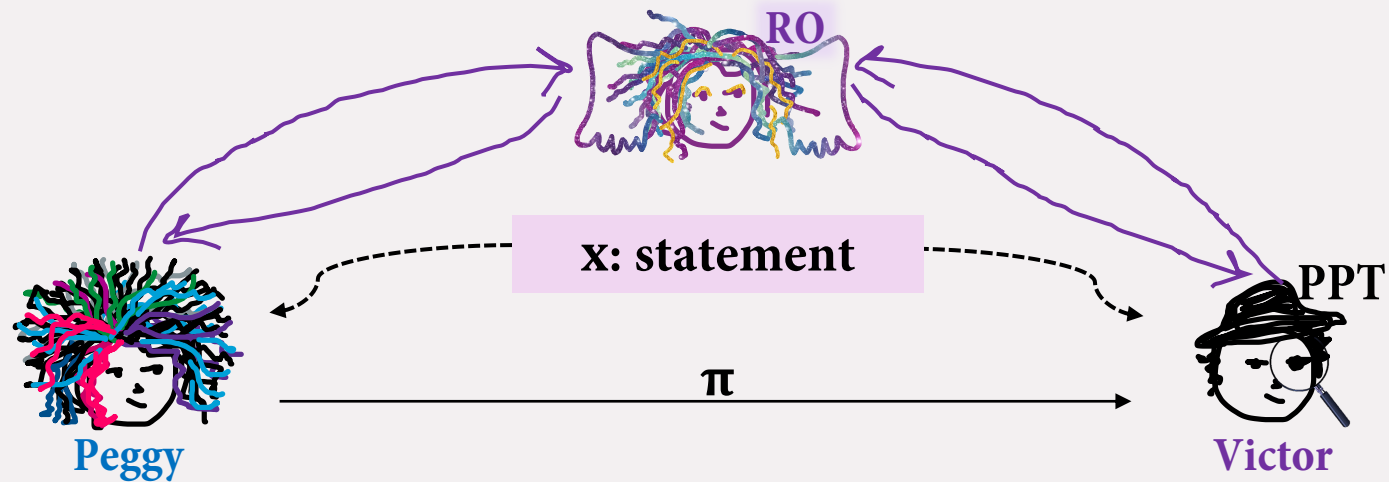
Not Sound!

Thm: If a language L has a non-interactive (one-message) ZK proof, then L can be solved in PPT.

*this is not possible as L_{G3C} is an NP-complete language!

Two ways to get Non-interactive ZK (NIZK)

1. Random Oracle Model and Fiat-Shamir Transform



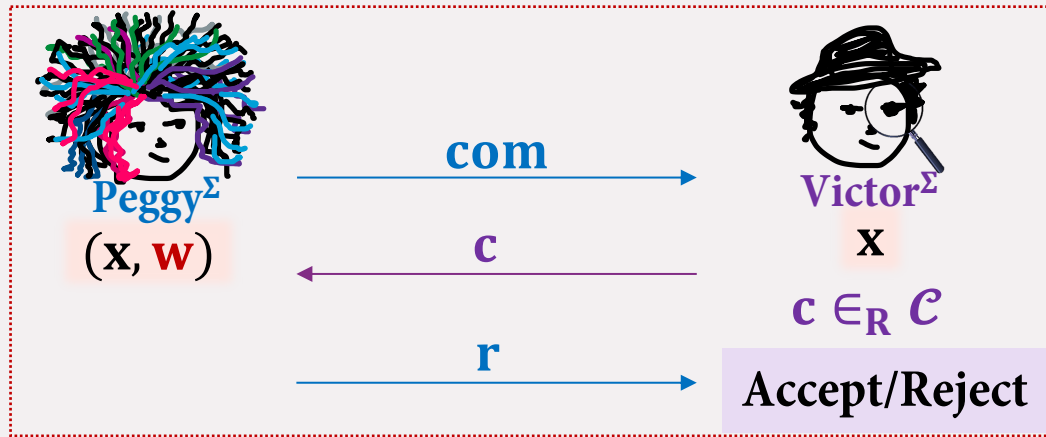
2. “Common Random String (CRS) Model”

(We won't see it in this course)

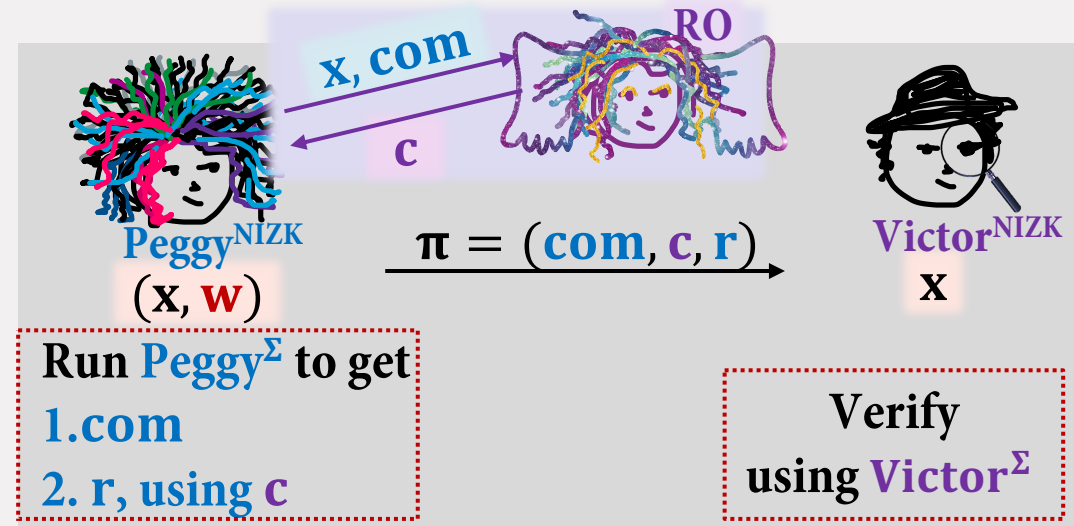
Fiat-Shamir Transform (Recall)



FIAT-SHAMIR: Replace the verifier with a random oracle.



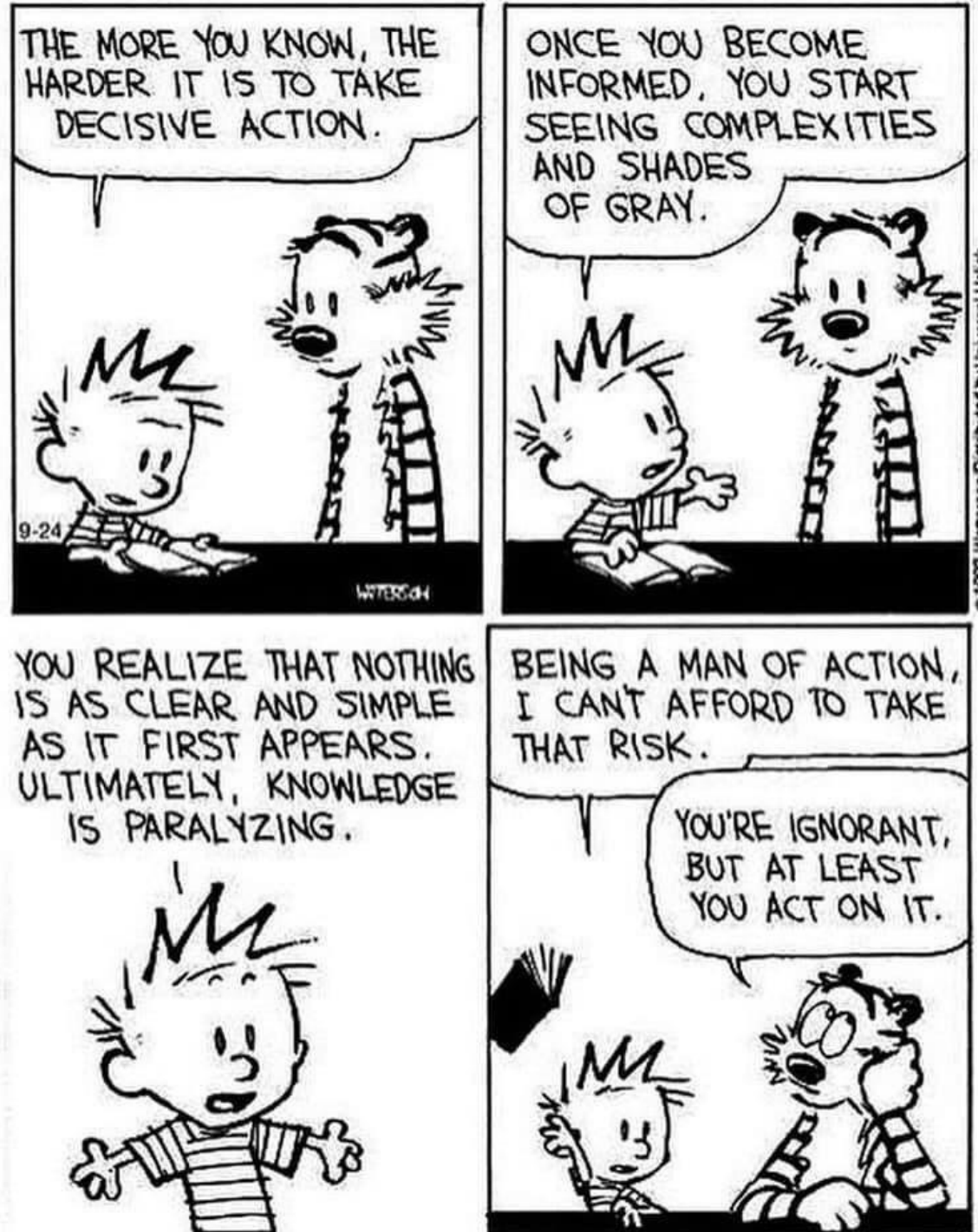
Σ -Protocol



NIZK Protocol: Fiat-Shamir

Proof is skipped!

THANK
YOU



Final Lab released!
Deadline: Wed, 29th Oct