PETS 25: Zero Knowledge

Florian Tramèr

Proof systems

- statements / languages

A language $L \subseteq \{0,1\}^*$

And a statement is "$x \in L$"

Ex: • $L = \{ pq \mid p, q$ are 1024 bit primes$\}$

stmt: $N = 128, \ldots, 321$ is in L

• $L = \{ x \mid Linux(x)$ crashes $\}$

↔ "all Linux bugs"

stmt: Linux crashes on input: 0x....

"One-shot proofs" / NP

∴ if verifier is poly time and deterministic, this is NP

What properties should a proof system have?

1. Completeness: If $x \in L$, then an honest prover convinces the verifier (w.p 100%)

2. Soundness: If $x \notin L$, then any prover P^* cannot convince the verifier

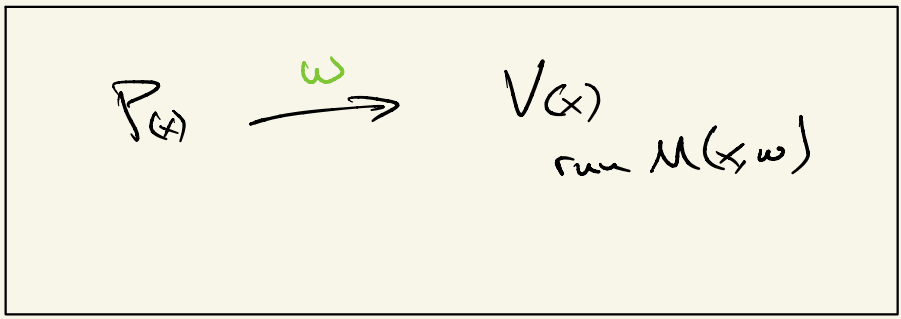
---

- if L has a "one shot" proof system where the verifier is deterministic (and efficient) and completeness/soundness hold absolutely → w.p 100% then $L \in NP$.

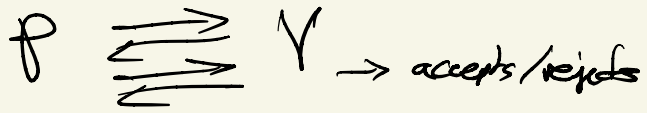
Def of NP A is in NP iff $\exists$ poly-time deterministic M (a verifier)

$$x \in L \iff \exists\, w \in \{0,1\}^{poly(|x|)} \text{ s.t } M(x,w)=1$$

(w: witness)



Why would we want interactive proofs?



Interactive proofs have many benefits:

1) Prove things outside of NP

Shamir '92: IP = PSPACE

(problems solvable w. polynomial space)

↑ languages w. interactive proofs

2) make shorter proofs

NP: proof of size $|w| = poly(|x|)$

SNARGs: proofs of size $O(\lambda)$, or $poly(\log(|x|))$

3) Zero knowledge: prove that $x \in L$ and nothing else!

Def: Interactive Proofs

a protocol between an unbounded prover and efficient (possibly randomized) verifier denoted as $\langle P, V \rangle(x) \in \{0,1\}$

(x: statement; $\langle P, V \rangle(x)$: the verifier's output)

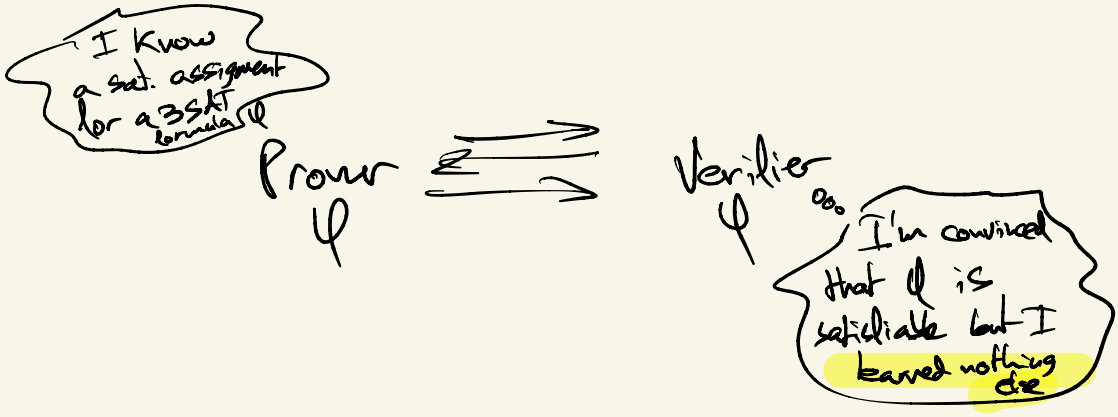
is an IP for a language L if it satisfies:

- Completeness: $\forall x \in L$, $\Pr_{P,V}[\langle P, V \rangle(x) = 1] \geq 2/3$

- Soundness: $\forall x \notin L$, $\forall$ possibly malicious P^*

$$\Pr_{V}[\langle P^*, V \rangle(x) = 1] \leq 1/3$$

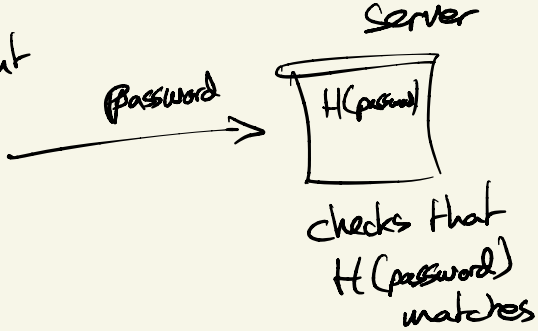
Zero Knowledge



Example applications

Passwords

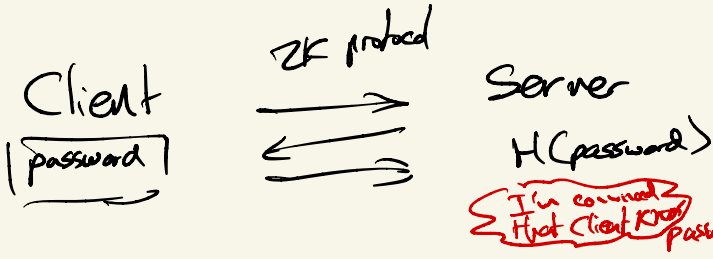
Real world:



Issues:

1) Server is hacked, Adv can observe password
2) Client can be phished

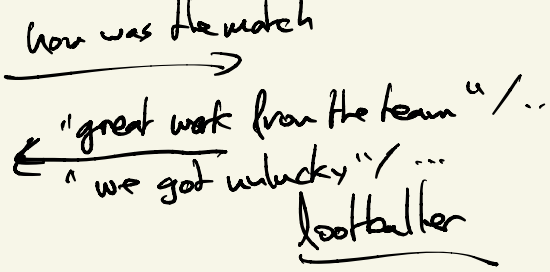
Better world:



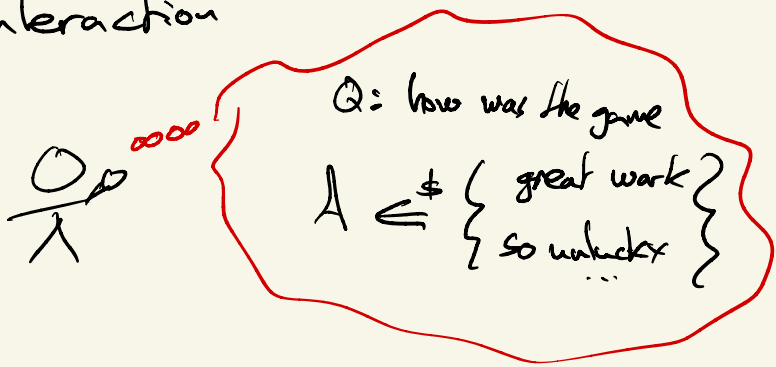
Defining ZK

What does it mean to "learn nothing" from an interaction?

Footballer interview



The interviewer could have simulated this entire interaction



Zero Knowledge = the simulated interaction is indistinguishable from the real one

Formal definition

An IP for a language L is ZK

if $\forall$ possibly malicious verifiers V^*,

$\exists$ a simulator Sim_{V^*} (efficient, randomized)

such that $\forall x \in L$:

$$View\{\langle P, V^* \rangle(x)\} \overset{c}{\approx} Sim_{V^*}(x)$$

- $View\{\langle P, V^* \rangle(x)\}$: everything the verifier sees
- $Sim_{V^*}(x)$: just the statement

Weird: if we can simulate a proof, how can we have soundness?

The simulator produces a valid transcript where the verifier accepts

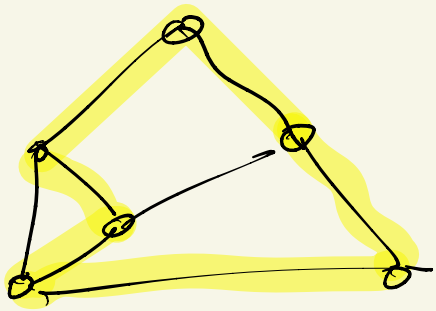
⚠ the simulator has more "power" than the prover

↳ the simulator doesn't have to follow the protocol order

An example

A ZK proof for Graph Hamiltonicity

A hamiltonian cycle visits every vertex once



$L = \{G : G \text{ has a ham. cycle}\}$

this is NP complete !

The protocol

Prover (G, cycle)

- pick a permutation Π over the vertices
- $\forall i,j \le n$ $B_{ij} = 1$ if $(\Pi(i), \Pi(j)) \in E$ else $B_{ij} = 0$

Verifier (G)

Commit(B_{11}) ... Commit(B_{nn})
Commit(Π)

$c \xleftarrow{R} \{0,1\}$

c

if $c=0$ open all commitments

if $c=1$ open only commitments to edges on the cycle

openings

Verifies everything is OK

if $c=0$: check that the committed Graph is isomorphic to G

if $c=1$: check that the opened commitments form a cycle

Thm: if we have a commit scheme that is perfectly binding and comp. hiding then this is a ZK proof system for Graph Hamiltonicity with:

- perfect completeness (100%)
- soundness error $1/2$
- computational ZK

By running the protocol λ times, we get $\text{negl}(\lambda)$ soundness error

---

Proof

- Completeness ✓ "obvious"
- Soundness: if the Graph does not have a hamiltonian cycle then the prover has to commit to either
  - a different graph
  - a graph w.o a cycle

  $\Rightarrow$ gets caught w.p. $\geq 50\%$

- Zero Knowledge:

we need to create a Simulator that outputs a view indist. from that of V^*

Sim(G):

1. Guess $\hat{c} \xleftarrow{\$} \{0,1\}$
   - if $\hat{c}=0$: commit to a perm. Π of G
   - if $\hat{c}=1$: commit to a complete graph
2. send the commitments to V^* who returns c
3. if $\hat{c} \neq c$: abort
4. else: open commitments for the verifier and output $(G, Comm, c, Open)$

- Claim 1: The Simulator reaches step 4 w.p $1/2 - neg(\lambda)$

proof: Commitments are hiding

- Claim 2: the output of the Simulator is comp. indist. from the verifier's view

Real View: Commit to $\Pi(G)$ ✓ → ← c ✓ → opening of $\Pi(G)$ or a cycle ✓

Sim View: Commit to $\Pi(G)$ or a complete graph → ← c ✓ → openings ✓

Amplifying Soundness

2 ways to "repeat" this protocol

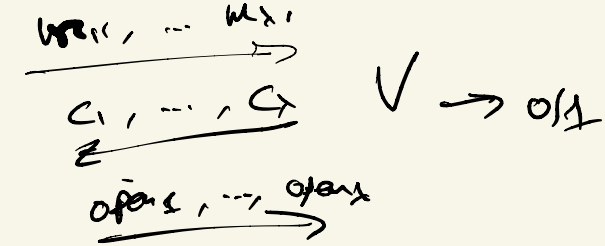
- In sequence:

$P \rightleftarrows V \rightarrow$ output 0/1

$P \rightleftarrows V \rightarrow$ output 0/1

⋮

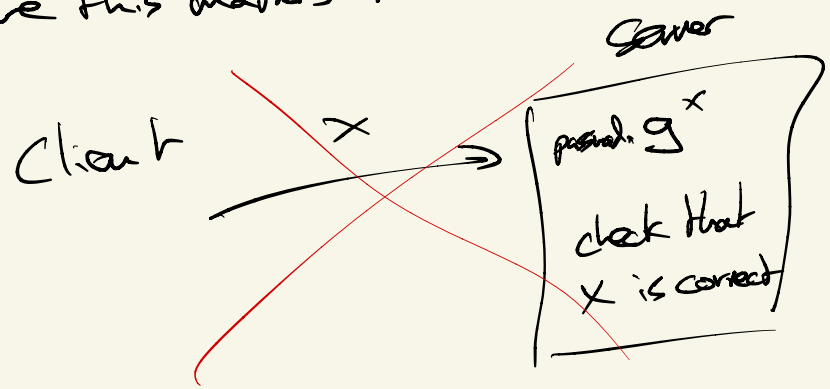
- In parallel:

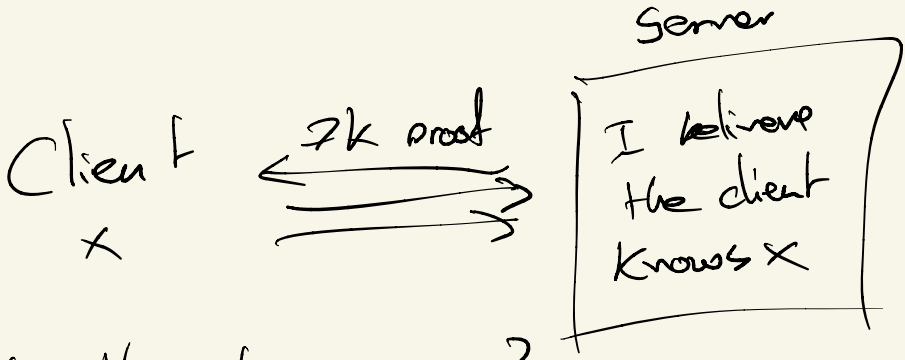


Proofs of knowledge

the protocol we saw proves that G has a Ham. cycle. Does this mean the prover "knows" the cycle?

Ex where this matters:





What's the language?

$$L = \{ h : \exists x \text{ s.t } g^x = h \}$$

Issue: $\forall h \in G, h \in L$

How do we define knowledge?

Through Torture

informal: an algorithm A "knows" a value x if you can recover x by interacting with the algorithm. ↳ potentially

Doesn't this contradict ZK?

this doesn't look like ZK

Def: Proof of Knowledge

A proof system $\langle P, V \rangle$ for an NP language L is a proof of knowledge (PoK) with knowledge error ε if there exists an efficient extractor Ext s.t. $\forall x, \forall P^*$

$$Pr\left[M(x,w)=1 : w \leftarrow Ext^{P^*}(x)\right] \geq Pr\left[\langle P^*, V\rangle(x)=1\right] - \varepsilon$$

How does the extractor work?

△ : it needs more "power" than the verifier

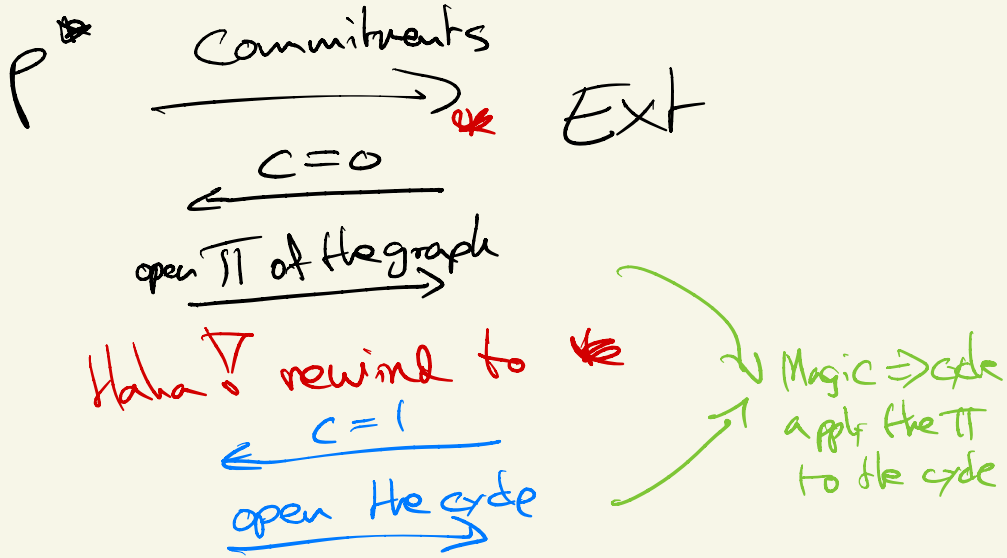
Idea: the extractor can rewind the prover

⇒ run the prover .... run the prover again

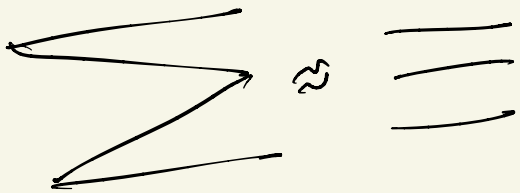
run the prover

save all state of the prover

Example for our Ham cycle ZK proof



Sigma Protocols



$P(x, w)$		$V(x)$
generate commitment u	$u \longrightarrow$	
		select challenge $c \xleftarrow{\$} C$
	$\longleftarrow c$	
generate response z	$z \longrightarrow$	
		outputs $b \leftarrow \text{verif}(x, u, c, z)$
		↳ deterministic

Properties

- perfect completeness
- "special" soundness: there exists an extractor Ext such that given two accepting transcripts (u, c, z), (u, c', z') for $c \neq c'$ it outputs a witness w s.t. $M(x, w) = 1$
- "Honest verifier" ZK: $\exists$ a simulator that takes input (x, c) where $x \in L$, $c \in C$ and outputs (u, c, z) s.t. $\text{View}(V) \overset{c}{\approx} \text{Sim}(\cdot)$

Thm: A sigma protocol for an NP language L is an interactive proof with soundness error at most $1/|C|$

Proof: From special soundness we know that *any* two valid transcripts $(u, c, z), (u, c', z')$ with $c \neq c'$ yield the witness

But if $x \notin L$, no witness exists $\forall$
So there can be *at most one* valid challenge for any commitment u

Since c is chosen at random, the prover can cheat w.p at most $1/|C|$

Notes:

→ Sim and Ext are just "proof artifacts" we never run these algos in practice

→ Sim^{V^*} can also run/rewind... the verifier V^*

Non-interactive ZK proofs

- Sigma protocol: 3 round $P \rightleftarrows V$
- Can we do one round? $P \xrightarrow{\pi} V$

This seems impossible! Why?

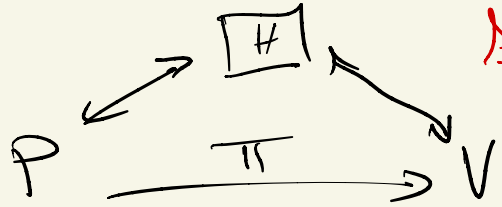
by ZK property, $\forall x \in L$ there exists a Sim such that $Sim(x) \approx_c \underbrace{View(V)}_{\pi}$

↳ $\exists$ an efficient, randomized algo for deciding the language $\Leftrightarrow L \in BPP$

The issue: the simulator and honest prover have the same power

$\Rightarrow$ in the "standard" model, NIZK is impossible

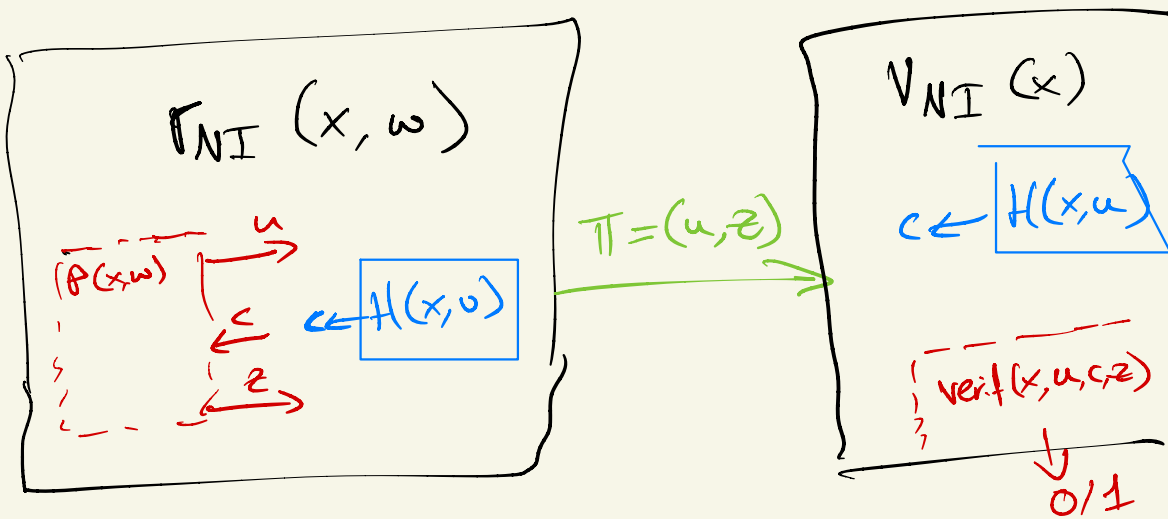
What about the Random Oracle Model?



$\therefore$ we'll allow the simulator to "program" the RO

The Fiat-Shamir transform

Idea: replace the verifier's challenge in the Sigma protocol by a RO query



Informal proof of security:

- completeness: ✓
- ZK: same thing as for the Σ protocol:
  Sim picks $c \xleftarrow{\$} C$ and runs
  $\text{Sim}_\Sigma(x,c) \to (u,c,z)$
  and it programs the RO s.t.
  $H(x,u) := c$
  and outputs $\pi = (u,z)$

- Proof of knowledge:

  Idea: run the prover until it queries $H(x,u)$. Return $c \in \mathcal{C}$. Get a valid transcript. Rewind and set $H(x,u)=c'$ for $c \neq c'$ and continue to get a different transcript

---

- A note on soundness: recall that for Σ protocols it was ok to have soundness error, say, $1/2$ because you can repeat! For a NIZK, there's nothing to repeat. So here it's important to start from a Σ protocol with negligible soundness error

- NIZK's in practice: