The Power of NAPs

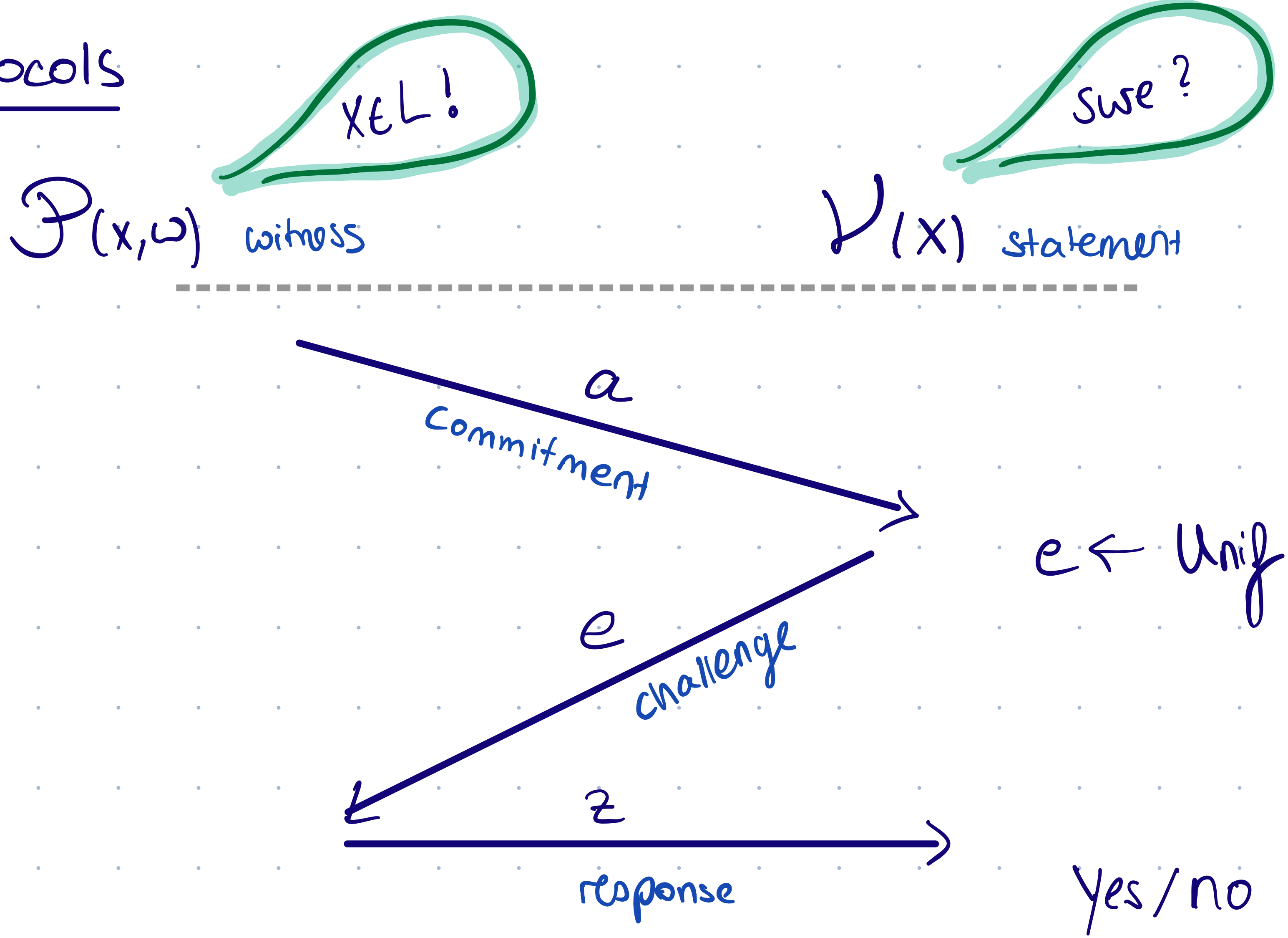
Compressing OR-proofs via Collision-Resistant Hashing

Séminaire ECO
21 janvier 2025
Montpellier

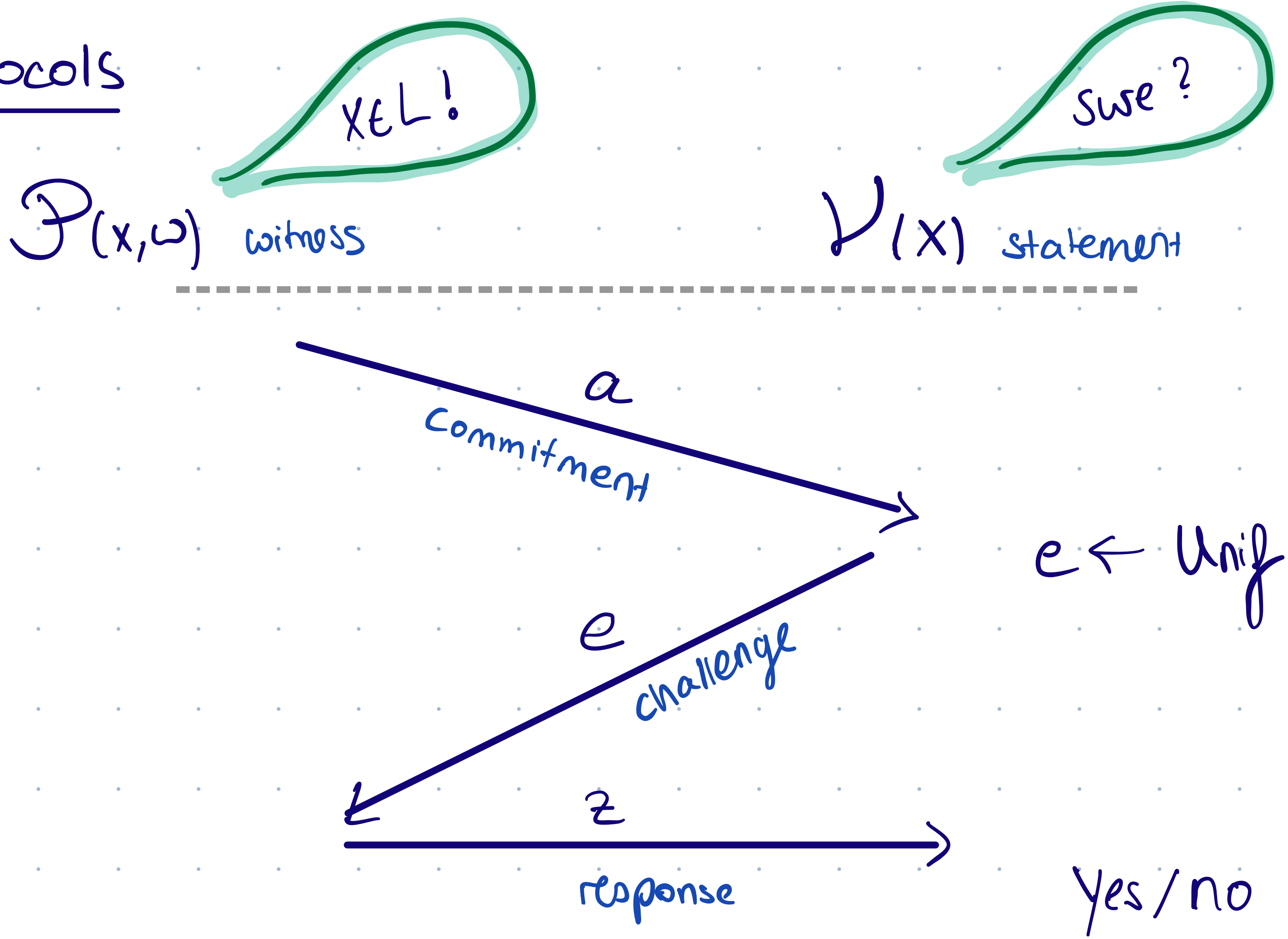
joint work with Mark Simkin

@TCC'24

Σ - Protocols



Σ - Protocols



Completeness ~ honest execution succeeds

Soundness ~ $\mathcal{P}$ must know witness ω to succeed

Honest-Verifier Zero-Knowledge (HVZK) ~ $\mathcal{V}$ doesn't learn ω

Honest-Verifier Zero-Knowledge

Real

$$\xrightarrow{\quad a \quad}$$
$$\xleftarrow{\quad e \quad}$$
$$\xrightarrow[\quad z \quad]{}$$

Simulated

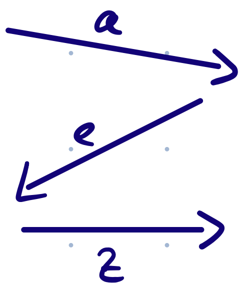
$e \leftarrow \mathrm{Unif}$

$(a, z) \leftarrow \mathrm{Simulator}(e)$

$$(a, e, z) \quad \approx \quad (a, e, z)$$

STRONG Honest-Verifier Zero-Knowledge

Real



Simulated

$e \leftarrow \text{Unif}$

$z \sim \text{Unif}$

$a := \text{Simulator}(e, z)$

(deterministic)

$$(a, e, z) \approx (a, e, z)$$

[Goel-Green-Hall-Andersen-Kaptchuk '22]

Σ-Protocol with HVZK $\Longrightarrow$ Σ-Protocol' with Strong HVZK

Σ-Protocol for Graph Isomorphism Problem

[Goldreich-Micali-Wigderson '86]

$\mathcal{P}(x, w)$ $\mathcal{V}(x)$

$x = (G_0, G_1)$

$w : \pi : G_0 \xrightarrow{\pi} G_1$

Σ-Protocol for Graph Isomorphism Problem

[Goldreich-Micali-Wigderson '86]

$\mathcal{P}(x,w)$ | $\mathcal{V}(x)$

$x = (G_0, G_1)$

$w: \pi:$

$$\begin{tikzcd} G_0 \arrow[r, "\pi"] \arrow[dr, "\tau"'] & G_1 \arrow[d, dashed] \\ & H \end{tikzcd}$$

$\xrightarrow{a = H}$

$e \leftarrow \{0,1\}$

$\xleftarrow{e}$

if $e=0$:
open left edge: τ

if $e=1$:
open right edge: $\tau \circ \pi^{-1}$

$\xrightarrow{z}$

check if $z(G_e) = a = H$

strong
Simulation: for e and z given, set $a := z(G_e)$

OR-Proofs

$\exists i \in [n] : x_i \in L$ $\quad\quad$ Sure ?

$\mathcal{P}(w, x_1, x_2, \dots, x_n)$ $\quad\quad$ $\mathcal{V}(x_1, x_2, \dots, x_n)$

$\xrightarrow{a}$

$\xleftarrow{e}$ $\quad\quad$ $e \leftarrow \text{Unif}$

$\xrightarrow{z}$

yes / no

Specific enough to be solved efficiently

General enough to be useful :

Ring Signatures, electronic voting

Constructing OR-Proofs

required communication overhead?

$\boxed{\Sigma\text{-Protocol}} \longrightarrow \boxed{\Sigma\text{-Protocol for OR-Proofs}}$

Constructing OR-Proofs

required communication overhead ?

Σ-Protocol $\longrightarrow$ Σ-Protocol for OR-Proofs

- [Cramer, Damgård, Schoenmakers 1994]: any Σ-Protocol
  but linear overhead
- From structured hardness assumptions (eg DLog, LWE,..)
  good communication complexity, but stronger assumptions
- via MPC-in-the-Head
  unstructured assumptions, but large communication complexity

Our Contributions

logarithmic communication overhead

Σ-Protocol
with strong HVZK
+
collision-resistant hashing

$\longrightarrow$

Σ-Protocol for OR-Proofs

+ new notion of non-adaptively programmable functions (NAPs)
+ rejection sampling can be explained

Any questions

so far?

Starting Point [Cramer, Damgård, Schoenmakers, 1994]

$(w_1, x_1) \in L$ $\qquad \mathcal{P}$ $\qquad (x_1, \dots, x_n)$ $\qquad \mathcal{V}$

honest: a_1

simulated: $e_2, \dots, e_n \leftarrow Unif$

$(a_i, z_i) \leftarrow Sim(e_i)$

$\xrightarrow{a_1, \dots, a_n}$

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

$\xleftarrow{e}$

e_1 st $e = e_1 + \dots + e_n$

honest: z_1

$$\xrightarrow[z_1, \dots, z_n]{e_1, \dots, e_n}$$

communication linear in n

check each (a_i, e_i, z_i) $\forall i \in [n]$

and $e = e_1 + \dots + e_n$

Our Ideas

$\mathcal{P}$ $\qquad\qquad\qquad\qquad$ $\mathcal{V}$

honest: a_1

simulated: $e_2, \dots, e_n \leftarrow$ Unif

$z_2, \dots, z_n \leftarrow$

$a_i :=$ StrongSim(e_i, z_i)

2 strong HVZK

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

$\xrightarrow{\cancel{a_1, \dots, a_n}}$

1 send hash

use seeds to compute

$e_1, \dots, e_n$

$z_1, \dots, z_n$

and Strong Simulator

to compute

$a_1, \dots, a_n$

$\xleftarrow{e}$

e_1 st $e = e_1 + \dots + e_n$

honest: z_1

$\xrightarrow[z_1, \dots, z_n]{e_1, \dots, e_n}$

check each

(a_i, e_i, z_i)

$\forall i \in [n]$

Our Ideas

$\mathcal{P}$ $\mathcal{V}$

honest: a_1

simulated: $e_2,\dots,e_n$ ← 3 Pseudorandom
$z_2,\dots,z_n$ ←

$a_i := \text{StrongSim}(e_i, z_i)$ 2 Strong HVZK

(a_2, e_2, z_2)
$\vdots$
(a_n, e_n, z_n)

$\xrightarrow{\cancel{a_1,\dots,a_n}}$ 1 send hash

$\xleftarrow{\;e\;}$

$\boxed{e_1}$ st $e = e_1 + \dots + e_n$

honest: $\boxed{z_1}$

3 send seeds

$\xrightarrow[\cancel{z_1,\dots,z_n}]{\cancel{e_1,\dots,e_n}}$

use seeds to compute
$e_1,\dots,e_n$
$z_1,\dots,z_n$
and Strong Simulator
to compute
$a_1,\dots,a_n$

check each
(a_i, e_i, z_i)
$\forall i \in [n]$

4 needs to program seeds

What we need is a function

- whose output looks random
- which for the same seed and same input is deterministic
- which can be privately programmed

- these exists notion of privately programmable PRF's but they require heavy tools (FHE, iO)

What we need is a function

- whose output looks random
- which for the same seed and same input is deterministic
- which can be privately programmed

- there exists notion of privately programmable PRF's but they require heavy tools (FHE, iO)

Key observation: we know programming location during key generation
$\Rightarrow$ MUCH SIMPLER

This is the moment where
we will use the
power of NAPs !

Non-Adaptively Programmable Functions

$seed \leftarrow Gen(i^* \in [n])$

$e_i \leftarrow Eval(seed, i)$

$pseed \leftarrow Prog(seed, e^*)$

$e_i \leftarrow PEval(pseed, i)$

Correctness:

1. programming worked

   $PEval(pseed, i^*) = e^*$

2. all other positions unchanged

   $PEval(pseed, i) = Eval(seed, i)$

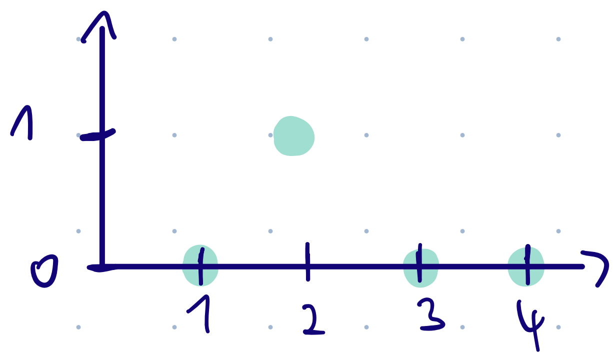
   $\forall i \neq i^*$

Private Programmability $\sim$ pseed does not leak the position i^*

Point Function

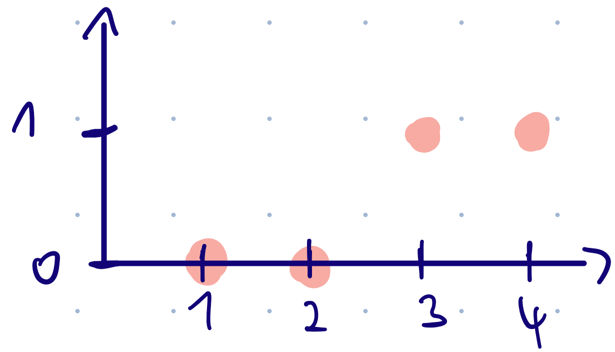
Distributed

$f(x)$



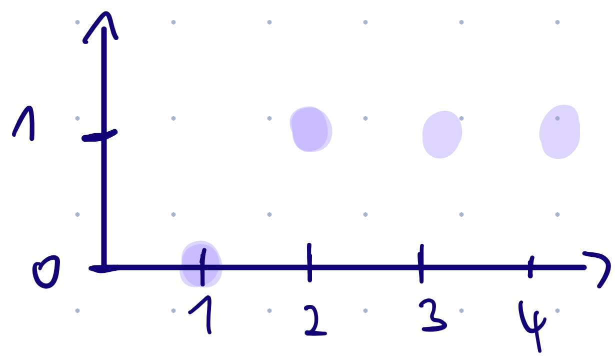
$=$

$f_0(x)$



$\oplus$

$f_1(x)$



$n=4$

[Boyle-Gilboa-Ishai'15]

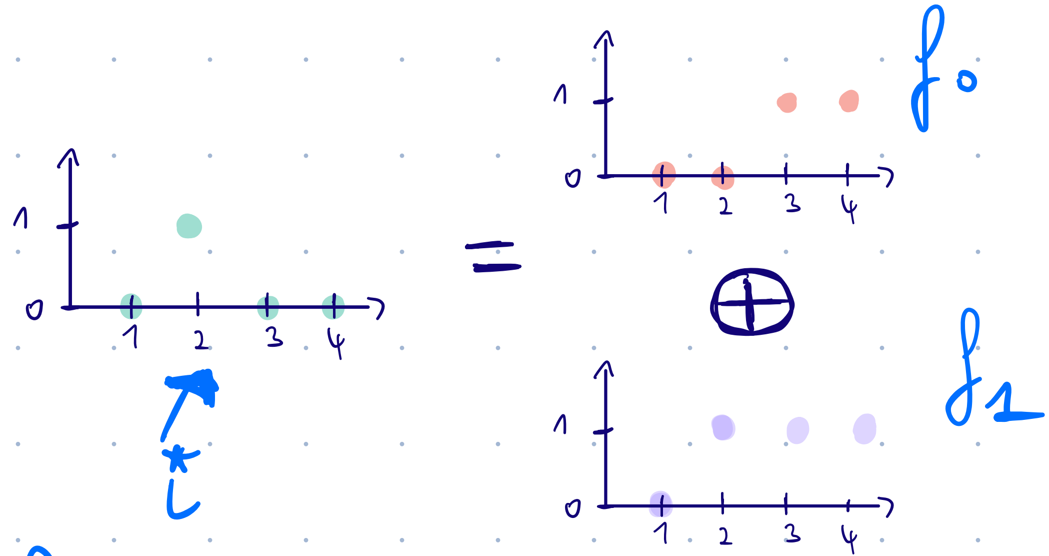
implied by collision-resistant hashing

Construncting NAPs via DPF

$n = 4$

$seed \leftarrow Gen(i^* \in [n])$

$pseed = (f_0, f_1, i^*)$



$e_i \leftarrow Eval(seed, i) = f_0(i)$

$pseed \leftarrow Prog(seed, e^*)$ either $f_0(i^*) = e^*$ or $f_1(i^*) = e^*$!

$e_i \leftarrow PEval(pseed, i) = f_{0/1}(i)$

for multiple output bits: concatenation

Our Construction

$\mathcal{P}$ $\qquad\qquad\qquad\qquad$ $\mathcal{V}$

honest: a_1

simulated: $e_2, \dots, e_n \leftarrow$ Eval

$z_2, \dots, z_n \leftarrow$

$a_i := \text{StrongSim}(e_i, z_i)$

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

$e_1, \dots, e_n \leftarrow$ PEval

$z_1, \dots, z_n \leftarrow$

$a_i := \text{StrongSim}(e_i, z_i)$

$\xrightarrow{H(a_1, \dots, a_n) = a}$

$\xleftarrow{e}$

$\boxed{e_1}$ st $e = e_1 + \dots + e_n$

honest: $\boxed{z_1}$

$\xrightarrow{pseed_e, pseed_z}$

$pseed_e \leftarrow \text{Prog}(e_1)$

$pseed_z \leftarrow \text{Prog}(z_1)$

check each

(a_i, e_i, z_i)

$\forall i \in [n]$

check $e = \Sigma e_i$

check $H(a_1, \dots, a_n) = a$

From 1-out-of-n to k-out-of-n

1-out-of-n: simulate $e_2, \dots, e_n$
obtain e

$\Big\}$ sum uniquely determines e_1

k-out-of-n: simulate $e_{k+1}, \dots, e_n$
obtain e

$\Big\}$ uniquely determine a polynomial $p(x)$ of degree $n-k$

$\Downarrow$

fixes $e_1, \dots, e_k$

$p(i) = e_i$

So... are we done?

Almost...

Σ-Protocol for Graph Isomorphism Problem

[Goldreich-Micali-Wigderson '86]

$\mathcal{P}(x,w)$ $\qquad\qquad$ $\mathcal{V}(x)$

$x = (G_0, G_1)$

$w: \pi:$

$$\begin{tikzcd} G_0 \arrow[r, "\pi"] \arrow[dr, "\tau"'] & G_1 \arrow[d, dashed] \\ & H \end{tikzcd}$$

$\xrightarrow{a = H}$

$e \leftarrow \{0,1\}$ ✓

$\xleftarrow{e}$

if $e = 0$:
open left edge: τ

if $e = 1$:
open right edge: $\tau \circ \pi^{-1}$

$\xrightarrow{z}$

check if

$z(G_e) = a = H$

z random isomorphism between graphs

More general distributions

(e.g. Rejection Sampling)

random coins

$\{0,1\}^\ell \ni r \leftarrow \mathrm{Eval}$

$z := \mathrm{Sample}(r)$

general distribution

But: for given $\boxed{z_1}$ need to find fitting r_1 s.t. $z_1 := \mathrm{Sample}(r_1)$

then program NAP to r_1

↳ Explainable Samplers [Lu-Waters '22]

Explainable Samplers

[Lu-Waters '22]

↳ showed that Rejection Sampling in the context of discrete Gaussians is explainable

→ we show that Rejection Sampling is explainable in general

Our Contributions

logarithmic communication overhead

Σ-Protocol with strong HVZK + collision-resistant hashing $\longrightarrow$ Σ-Protocol for OR-Proofs

+ new notion of non-adaptively programmable functions (NAPs)
+ rejection sampling can be explained

Thank you :)