The Power of NAPs

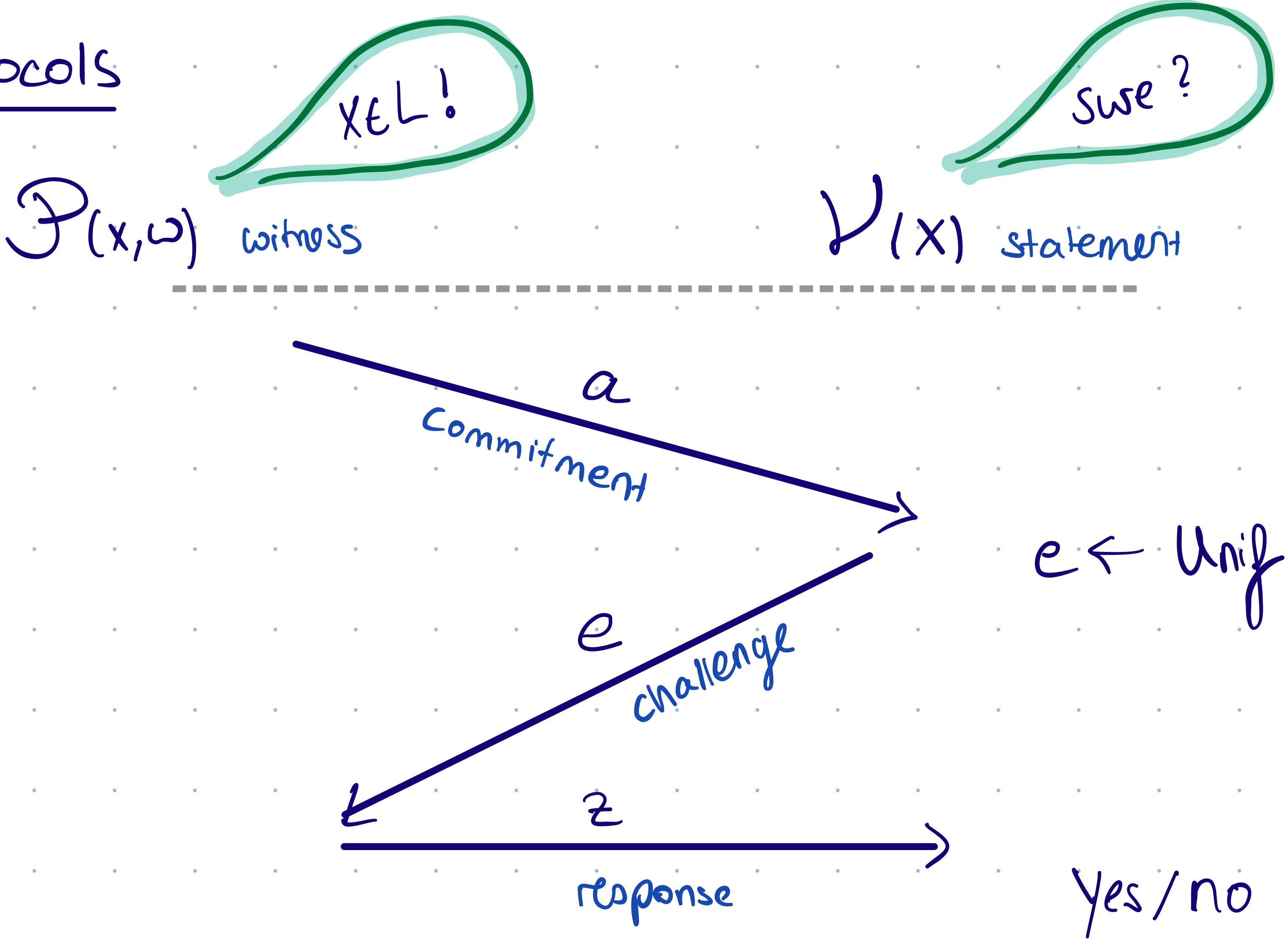
Compressing OR-proofs via Collision-Resistant Hashing

Séminaire C2
17 janvier 2025
Nancy

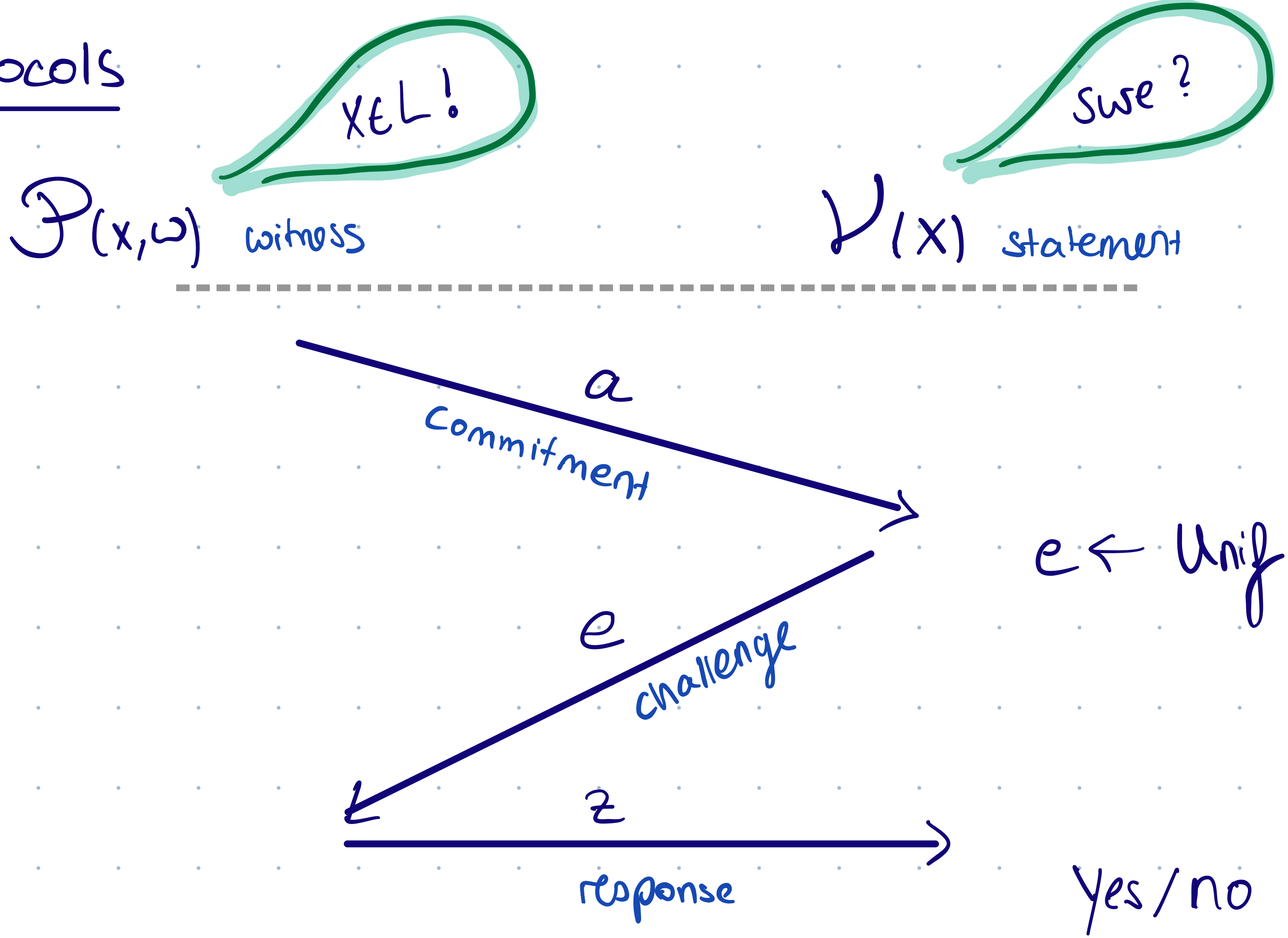
joint work with Mark Simkin

@TCC'24

Σ - Protocols



Σ - Protocols



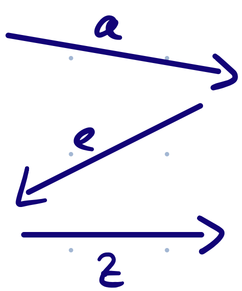
Completeness ~ honest execution succeeds

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



Simulated

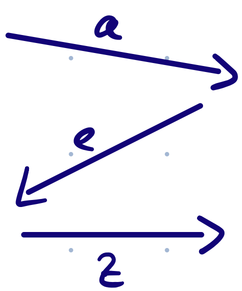
$e \leftarrow \mathrm{Unif}$

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

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

STRONG Honest-Verifier Zero-Knowledge

Real



Simulated

$e \leftarrow$ Unif

$z \sim$ Unif

$a :=$ 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]

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

$x = (G_0, G_1)$

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

Σ-Protocol for Graph Isomorphism Problem

[Goldreich-Micali-Wigderson '86]

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

$x = (G_0, G_1)$

$w: \pi:$

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

$\xrightarrow{a = H}$

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

$\xleftarrow{e}$

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

$\exists i \in [n] : x_i \in L$

Sure ?

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

$\xrightarrow{\quad a \quad}$

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

$\xrightarrow{\quad z \quad}$ $\qquad$ yes / no

Specific enough to be solved efficiently

General enough to be useful : Ring Signatures, electronic voting

Constructing OR-Proofs

required communication overhead ?

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

Constructing OR-Proofs

required communication overhead ?

Σ-Protocol → 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

→

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]

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

honest: a_1

simulated: $e_2, \dots, e_n \leftarrow \mathrm{Unif}$

$(a_i, z_i) \leftarrow \mathrm{Sim}(e_i)$

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

(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

$$\underbrace{\xrightarrow[z_1, \dots, z_n]{e_1, \dots, e_n}}_{\text{communication linear in } n}$$

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

and $e = e_1 + \dots + e_n$

Our Ideas

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

honest: a_1

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

$z_2, \dots, z_n \leftarrow$ Unif

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

② strong HVZK

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

① send hash

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

$\xleftarrow{e}$

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

honest: z_1

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

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)

$\forall i \in [n]$

Our Ideas

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

honest: a_1

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

$z_2, \dots, z_n$ ←

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

② strong HVZK

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

~~$a_1, \dots, a_n$~~ ① send hash

e

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

③ send seeds

~~$e_1, \dots, e_n$~~

~~$z_1, \dots, z_n$~~

honest: $\boxed{z_1}$

④ needs to program seeds

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)

$\forall i \in [n]$

<u>What we need</u> is a function

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

- there 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

- these 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

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

$e_i \leftarrow Eval(msk, i)$

$psk \leftarrow Prog(msk, e^*)$

$e_i \leftarrow PEval(psk, i)$

Correctness:

1. programming worked

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

2. all other positions unchanged

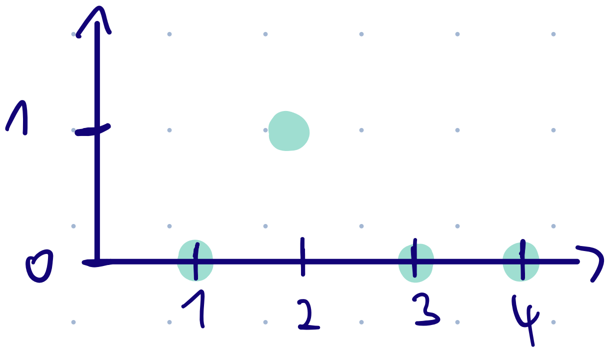
   $PEval(psk, i) = Eval(msk, i)$

   $\forall i \neq i^*$

Private Programmability ~ psk does not leak the position i^*

Point Function

$f(x)$

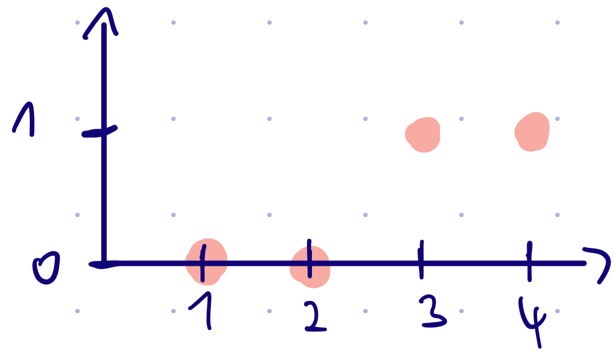


$n=4$

$=$

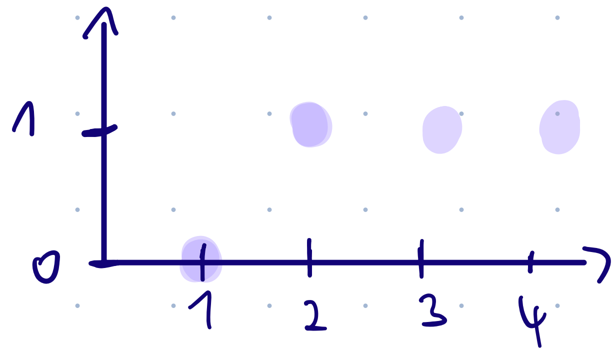
Distributed

$f_0(x)$



$\oplus$

$f_1(x)$



[Boyle-Gilboa-Ishai '15]

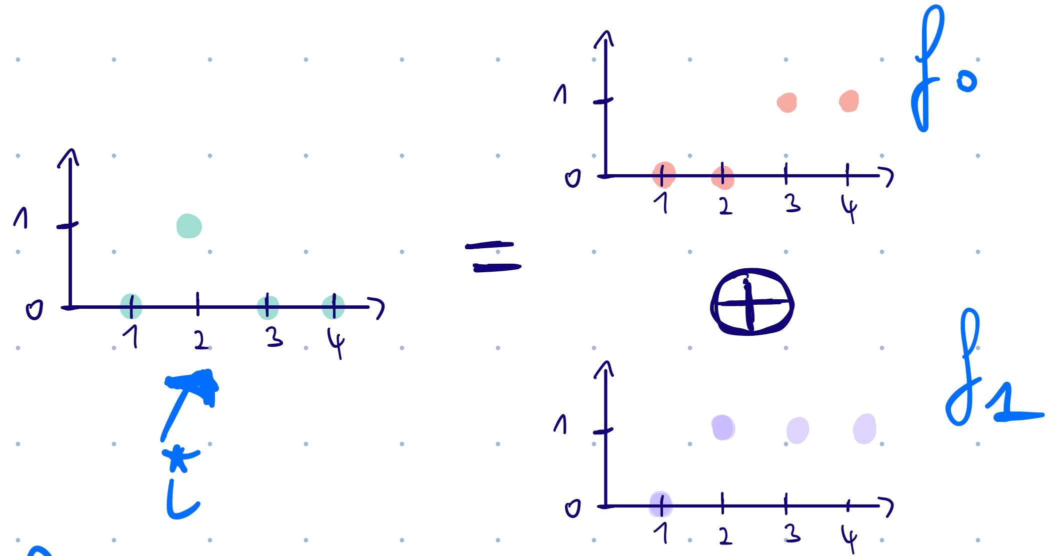
implied by collision-resistant hashing

Construncting NAPs via DPF

$n=4$

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

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



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

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

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

for multiple output bits: concatenation

Our Construction

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

honest: a_1

simulated: $e_2, \dots, e_n$, $z_2, \dots, z_n \leftarrow$ Eval

$e_1, \dots, e_n$, $z_1, \dots, z_n \leftarrow$ PEval

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

(a_2, e_2, z_2)

$\vdots$

(a_n, e_n, z_n)

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

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

$$\xleftarrow{\quad e \quad}$$

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

honest: $\boxed{z_1}$

$psk_e \leftarrow \mathrm{Prog}(e_1)$

$psk_z \leftarrow \mathrm{Prog}(z_1)$

$$\xrightarrow{psk_e,\ psk_z}$$

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 $e_2, \ldots, e_n$ } sum uniquely determines e_1
obtain e

k-out-of-n: simulate $e_{k+1}, \ldots, e_n$ } uniquely determine a polynomial $p(x)$ of degree $n-k$
obtain e

$\Downarrow$

fixes $e_1, \ldots, e_k$

$p(i) = e_i$

So... are we done?

Almost...

Σ-Protocol for Graph Isomorphism Problem

[Goldreich-Micali-Wigderson '86]

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

$x = (G_0, G_1)$

$w: \pi:$

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

$\xrightarrow{\quad a = H \quad}$

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

$\xleftarrow{\quad e \quad}$

if $e = 0$:
open left edge: τ

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

$\xrightarrow{\quad z \quad}$

check if

$z(G_e) = a = H$

z random isomorphism between graphs

More general distributions

- use uniform bits as the random coins for sampling other distributions

  (e.g. Rejection Sampling)

- use a NAP to program the random bits
- need a way to go from
  - → random bits to a sample (easy)
  - → a sample to "fitting" random bits (difficult)

⟹ 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 → Σ-Protocol for OR-Proofs

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

Thank you :)