Fiat-Shamir via List-Recoverable Codes

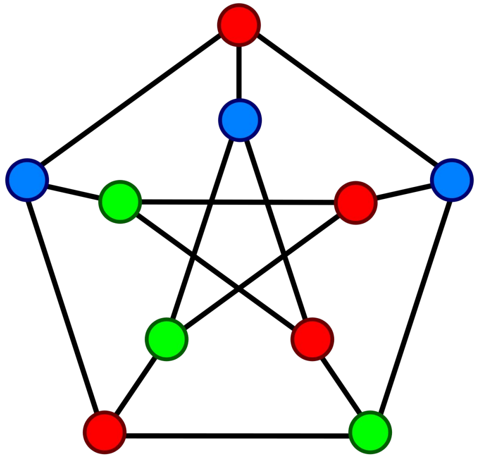
Alex Lombardi
MIT

Joint work with Justin Holmgren (NTT Research) and Ron Rothblum (Technion)

Abstract

- We soundly instantiate the Fiat-Shamir heuristic for a broad class of protocols
  - E.g. parallel repetitions of all "commit-and-open" protocols
- Leverage a new connection to list-recoverable codes.
  - New kind of derandomized parallel repetition

Zero Knowledge for NP [GMW86]



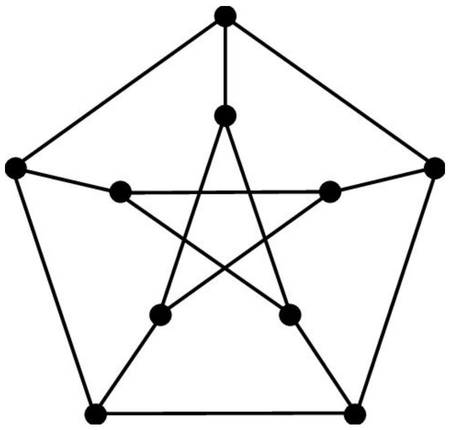
$P(G$

$\pi \in_R S_3$

$\alpha = \text{Commit}(\pi \circ \chi)$ →

← β

$\gamma = \text{Open}(\pi(\chi(v)), \pi(\chi(w)))$ →



$V(G$

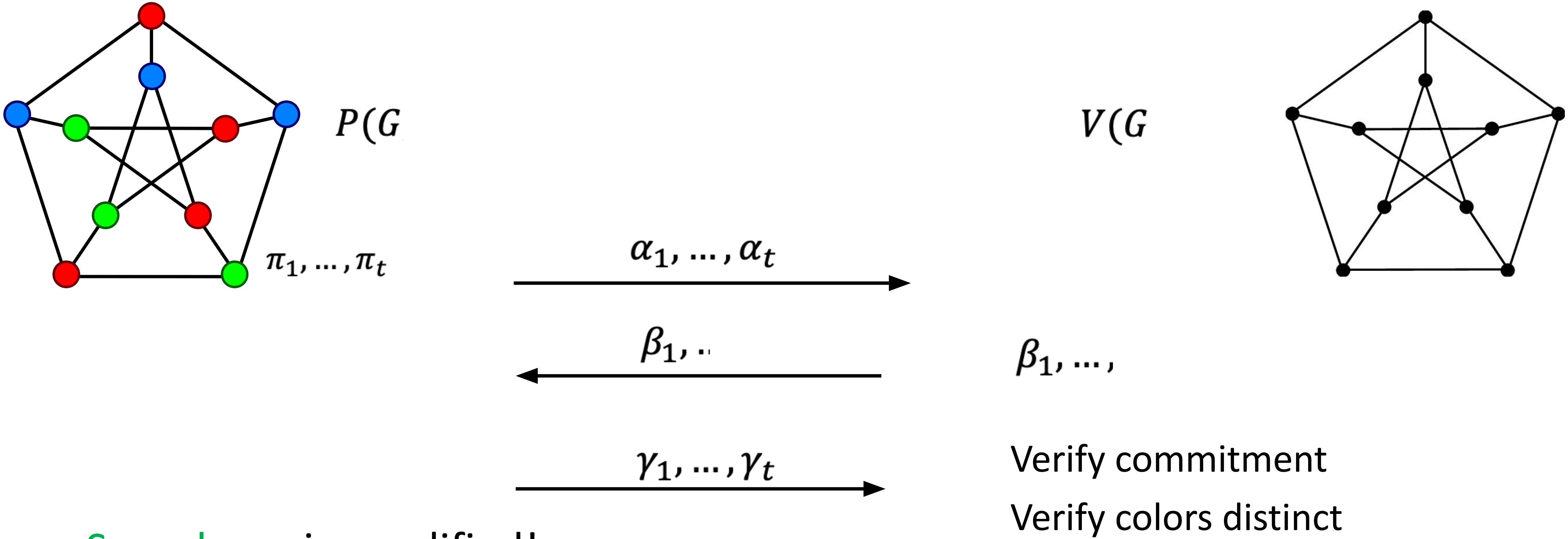
$\beta = ($

Verify commitment

Verify colors distinct

- **Soundness Error** $1 - \frac{1}{|E|}$
- Improve soundness error (to negligible) via **sequential repetition**, preserving ZK

How about Parallel Repetition?



- Soundness is amplified!
- **Open problem: is this ZK?**

This Work: No (for a natural Com in the CRS model), assuming LWE

The Fiat-Shamir Transform [FS86]

Public-Coin Interactive Protocol → *generically* → Non-Interactive Argument

Public-Coin Interactive Protocol

$\boldsymbol{P}$ — $\boldsymbol{V}$

- $P \to V$: α_1
- $V \to P$: β_1
- …
- $P \to V$: α_{r-1}
- $V \to P$: β_{r-1}
- $P \to V$: α_r

(Each β_i uniformly random)

Non-Interactive Argument

Hash Function h

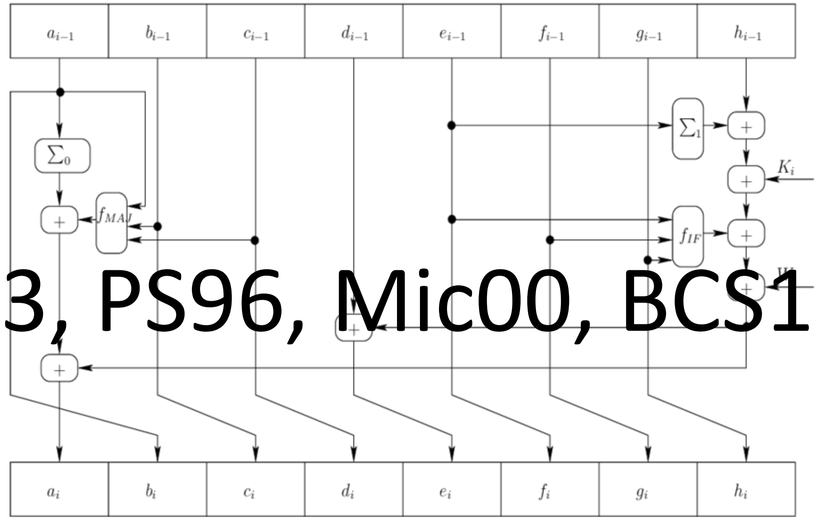
$\boldsymbol{P_{FS}}$ — $\boldsymbol{V}$

- $P_{FS} \to V$: $\alpha_1, \ldots, \alpha_r$

$$\beta_1 = h(x, \alpha_1)$$
$$\beta_2 = h(x, \alpha_1, \alpha_2)$$
…
$$\beta_i = h(x, \alpha_1, \ldots, \alpha_i)$$

Heuristically (and in practice), soundness is preserved.

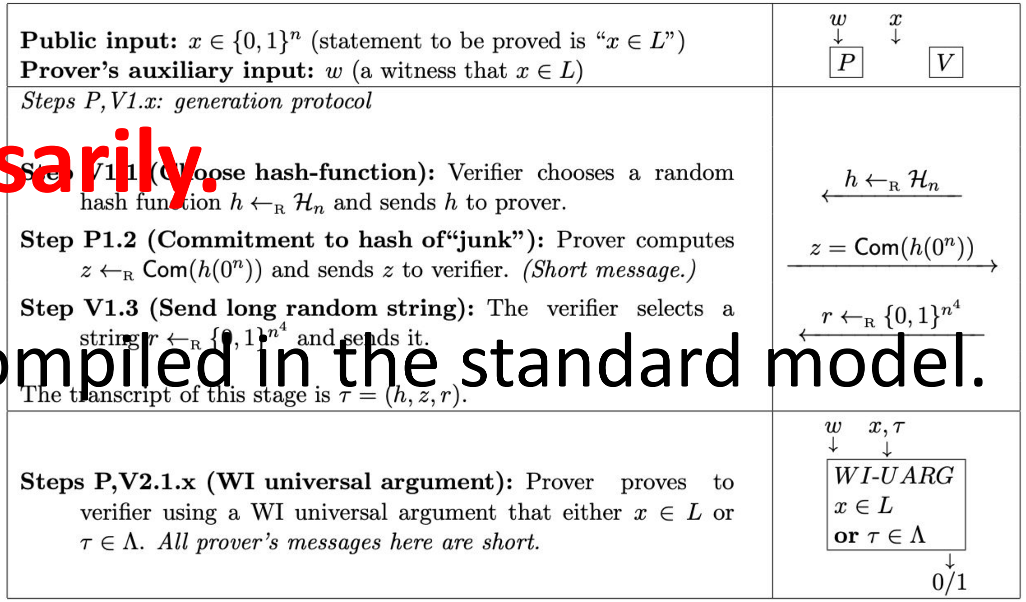
Is Fiat-Shamir secure?



[BR93, PS96, Mic00, BCS16]: **Yes, in the random oracle model.**

[Bar01, GK03, BBHMR19]: **Not necessarily.**

Some interactive arguments **cannot** be compiled in the standard model.



Public input: $x \in \{0,1\}^n$ (statement to be proved is "$x \in L$")
Prover's auxiliary input: w (a witness that $x \in L$)

Steps P, V1.x: generation protocol

Step V1.1 (Choose hash-function): Verifier chooses a random hash function $h \leftarrow_R \mathcal{H}_n$ and sends h to prover.

Step P1.2 (Commitment to hash of "junk"): Prover computes $z \leftarrow_R \mathsf{Com}(h(0^n))$ and sends z to verifier. *(Short message.)*

Step V1.3 (Send long random string): The verifier selects a string $r \leftarrow_R \{0,1\}^{n^4}$ and sends it.

The [illegible] of this stage is $\tau = (h, z, r)$.

Steps P,V2.1.x (WI universal argument): Prover proves to verifier using a WI universal argument that either $x \in L$ or $\tau \in \Lambda$. *All prover's messages here are short.*

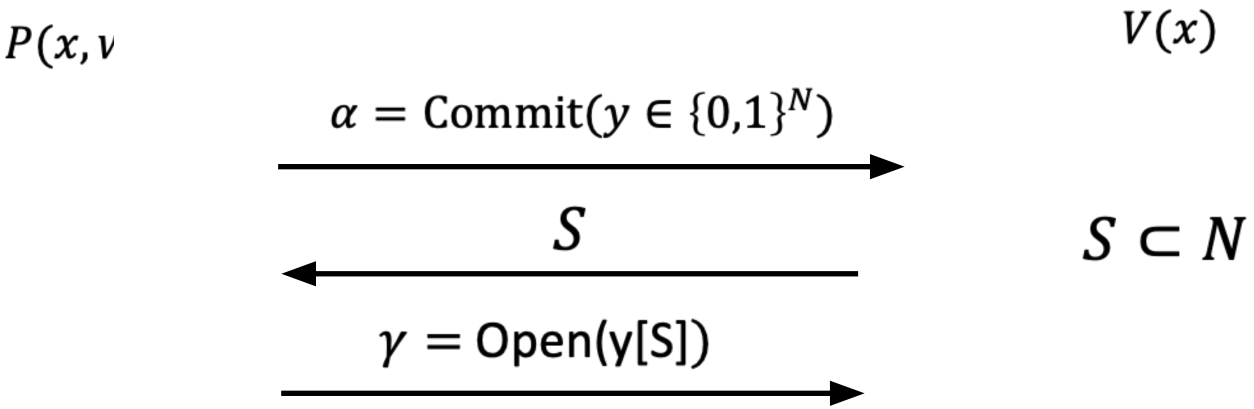
Is Fiat-Shamir secure?

Our Goal: Establish a stronger theoretical basis for this transformation

[KRR16, CCRR18, HL18, CCHLRRW19, PS19, LVW19, GJJM19, BFJKS19, LNPT19, LV20a, BKM20, JKKZ20, CLMQ20, LNPY20, LV20b, HLR21, …]

Our Results

1) Under the **LWE** assumption, Fiat-Shamir can be instantiated for (the parallel repetition of) **any** commit-and-open protocol (e.g. GMW 3-coloring)



- Every such protocol has a **NIZK variant**! (e.g. non-interactive MPC-in-the-head)
- Every such protocol is **not ZK** [DNRS99]

2) (Informal) FS for any protocol with ``efficiently recognizable bad challenges.” Prior work needed “efficiently enumerable bad challenges,” which is much more restrictive.

Main Takeaways

1) Much more widely applicable FS instantiation.

2) Resolve 35 year old intro crypto problem.

3) Cool new connection to coding theory/derandomization!

Correlation Intractability

[CGH04]

A hash family H is **correlation intractable** for a (sparse) relation R if:

$\forall$PPT A,

$$\Pr_{\substack{h \leftarrow H \\ x \leftarrow A(h)}} \left[\left(x, h(x)\right) \in R \right] = \text{negl}$$

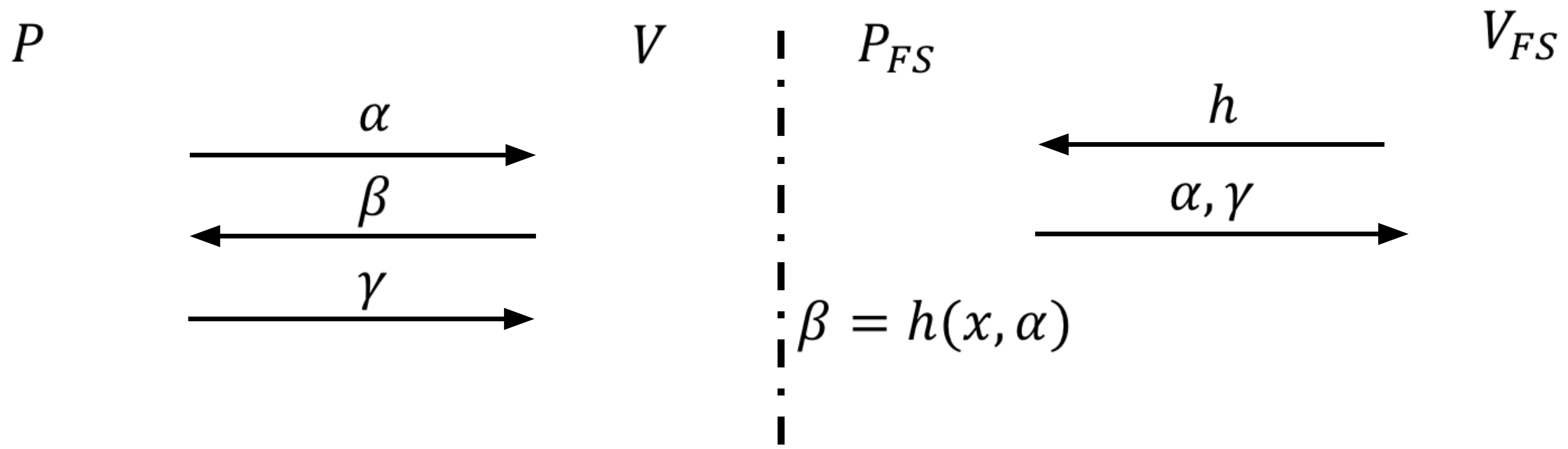
Theorem [CCHLRRW19, PS19]: under standard assumptions, there exists a hash family H that is CI for all **functions** computable in time T.

- $h \in H$ can be evaluated in time $T \cdot \text{poly}(\lambda)$

The Bad-Challenge Function Paradigm

[CCHLRRW19]

Public-Coin Interactive Proof Π ➡ Non-Interactive Argument Π_{FS}



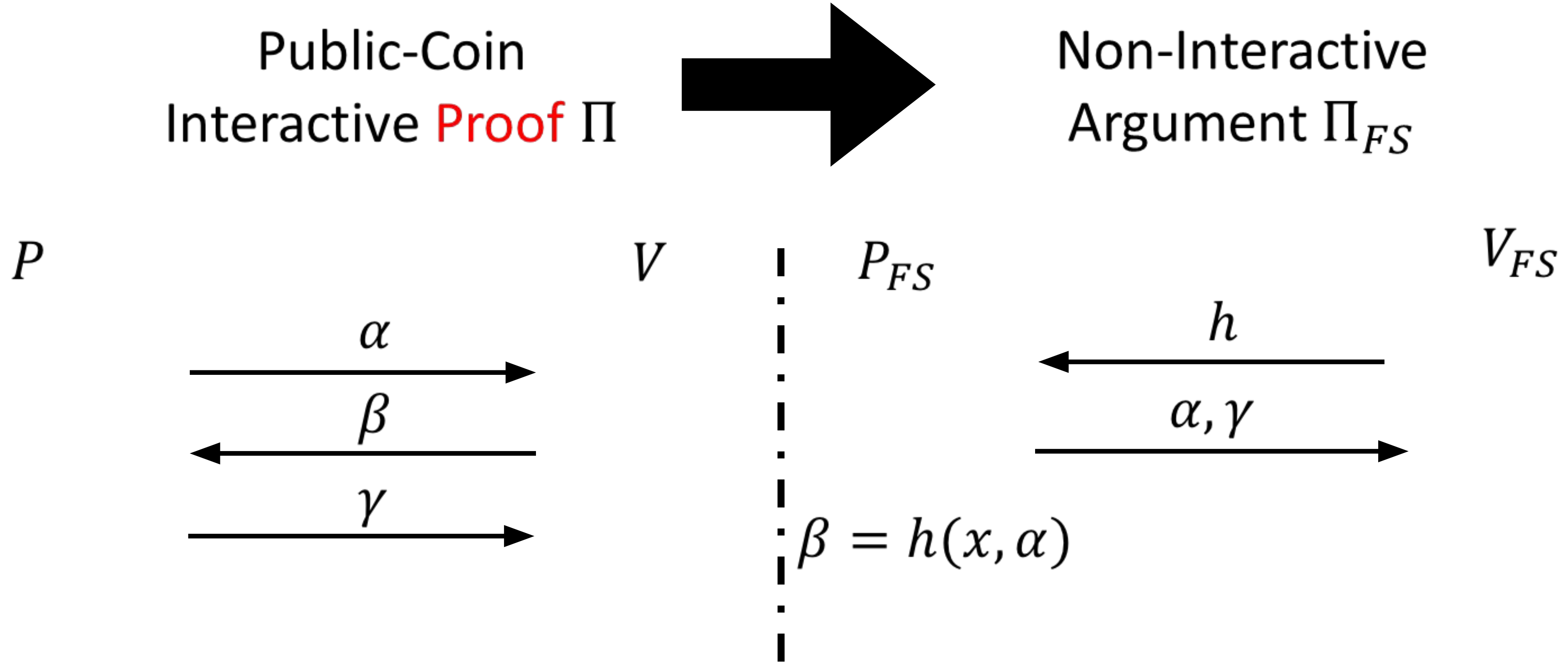
Suppose that for all $x \notin L$ and all α, $\exists$ at most one β s.t. V accepts $(x, \alpha, \beta, \gamma)$

Let $f(x, \alpha) = \beta^*$ be the bad-challenge function for Π

If $\mathcal{H}$ is CI for f, then Π_{FS} is sound!

If f is efficiently computable, $\exists$ such $\mathcal{H}$!

What if there are many bad challenges?



Suppose that for all $x \notin L$ and all α, $\exists$ at most B bad choices of β

Let $f_i(x, \alpha) = \beta_i^*$ be the ith bad-challenge function for Π

If $\mathcal{H}$ is CI for a random f_i, then Π_{FS} is sound! Security loss: $\frac{1}{B}$

The Problem

Can we handle protocols that have **many bad challenges?**

Can we construct hash functions that are CI for **relations** that are not functions?

The Solution

Can we handle protocols that have **many bad challenges?**

Can we construct hash functions that are CI for **relations** that are not functions?

Yes!

(when the relations have nice structure)

Product Relations

$$R = \{(x, (y_1, \ldots, y_t))\} \subset \{0,1\}^n \times (\{0,1\}^m)^t$$

Definition: R is a product relation if for all inputs x,

$$R_x = S_1 \times S_2 \times \cdots \times S_t$$

for some sets $S_1, \ldots, S_t \subset \{0,1\}^m$

Product relations may have **many bad points**, but they have **combinatorial structure**.

Product Relations

Definition: R is a product relation if for all inputs x,

$$R_x = S_1 \times S_2 \times \cdots \times S_t$$

for some sets $S_1, \ldots, S_t \subset \{0,1\}^m$

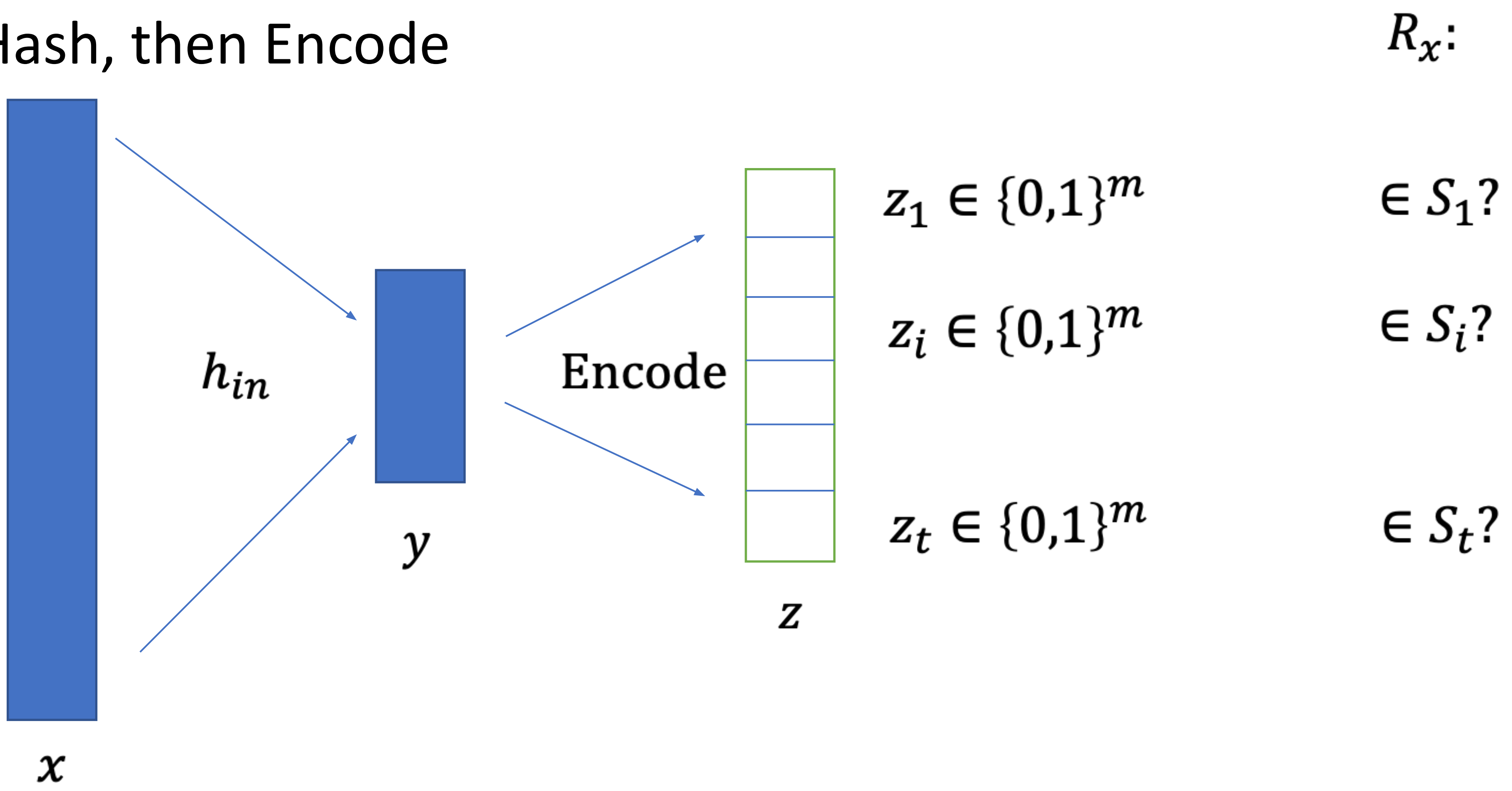
Main Theorem: Under LWE, there exist CI hash functions for product relations*

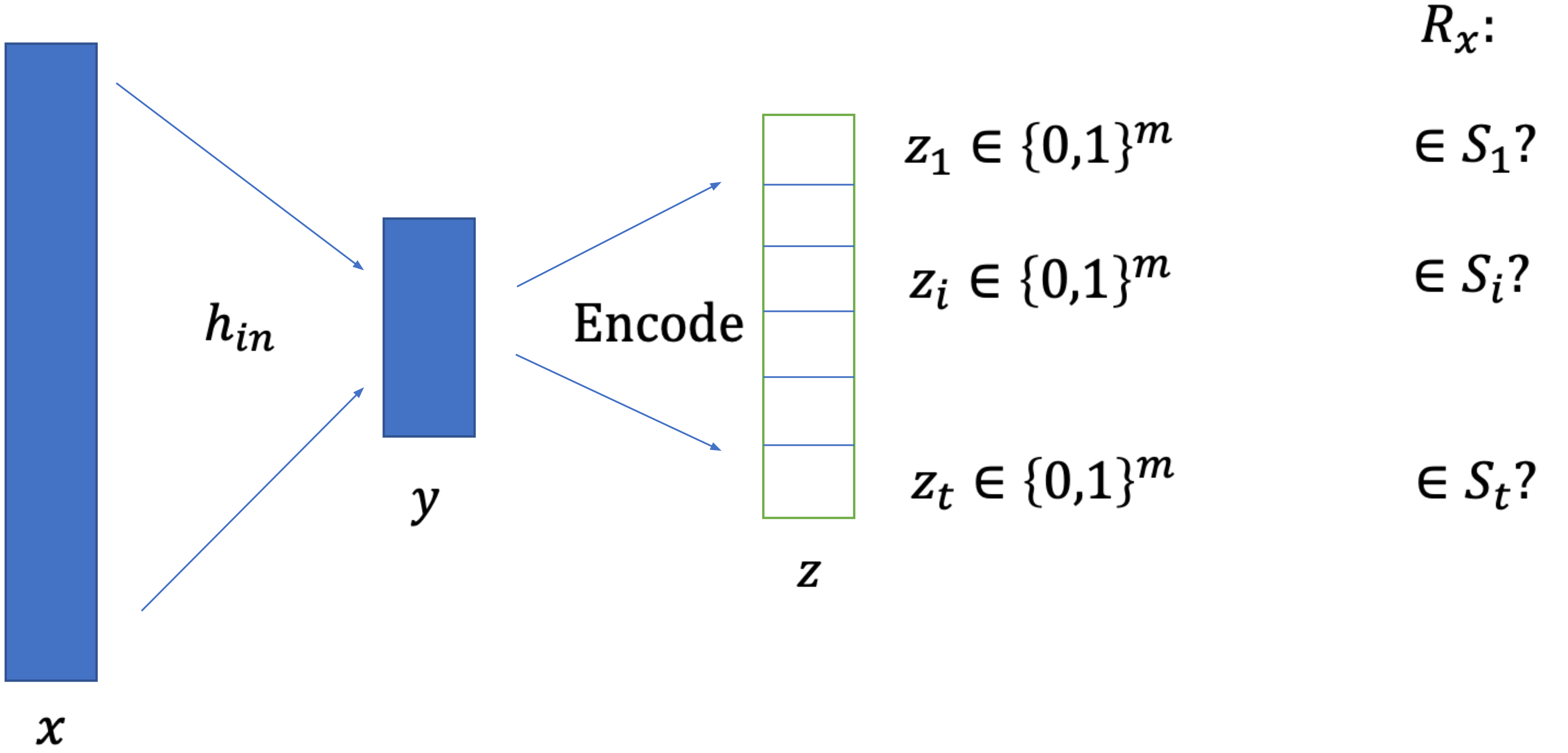
*The "repetition parameter" t needs to be large enough, depending on the density of the S_i
*We need membership in S_i to be efficiently decidable

CI for Product Relations

Main Theorem: Under LWE, there exist CI hash functions for product relations*

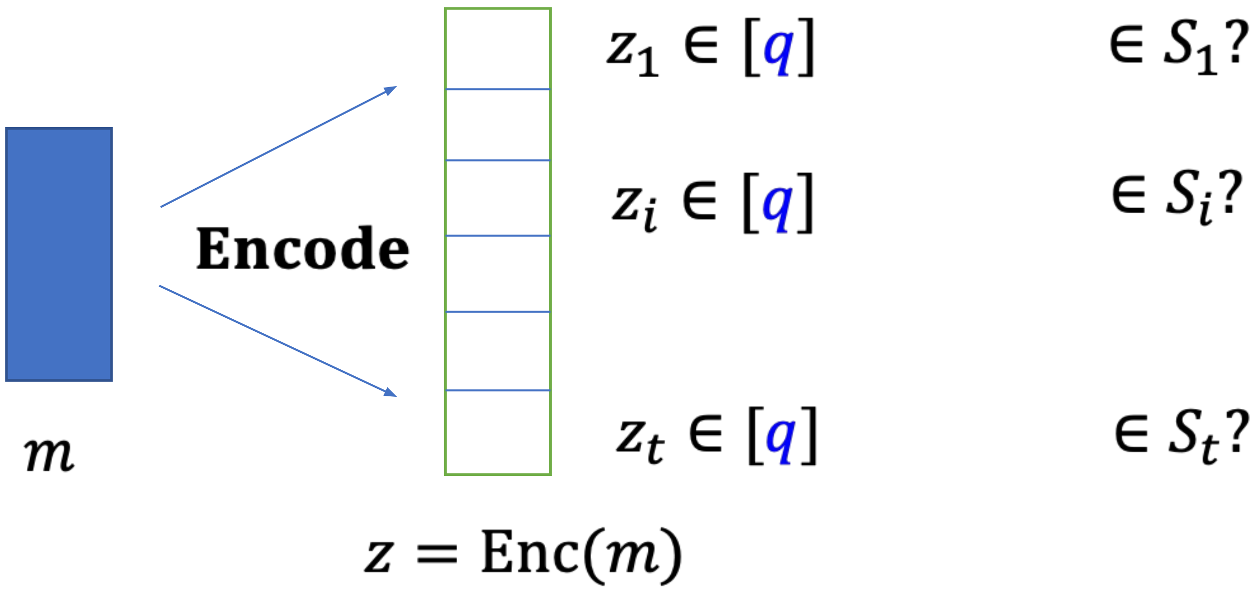
Idea: Hash, then Encode

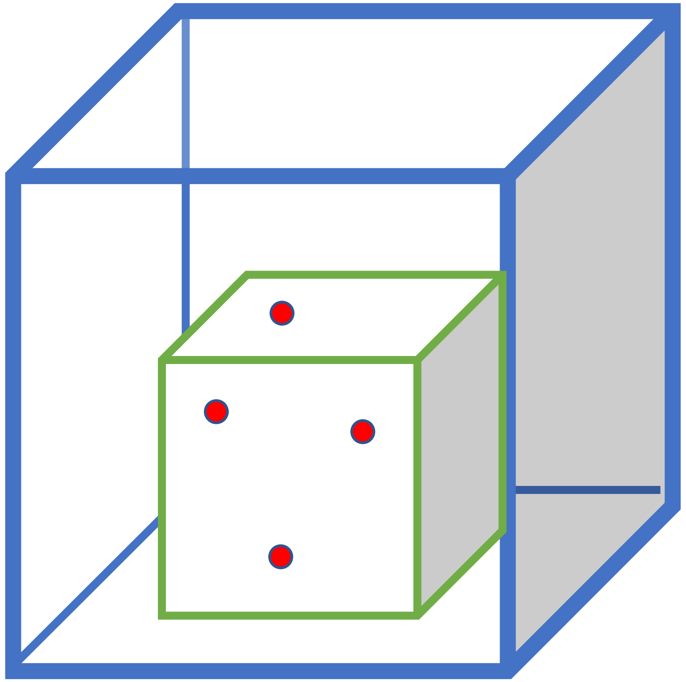




- Reduce the number of bad points
  - For every x, there may be many bad z, but hopefully few bad y (and so few bad z in the image of the hash function.
  - Use the [PS19] hash function for h_{in}

Codes to the Rescue



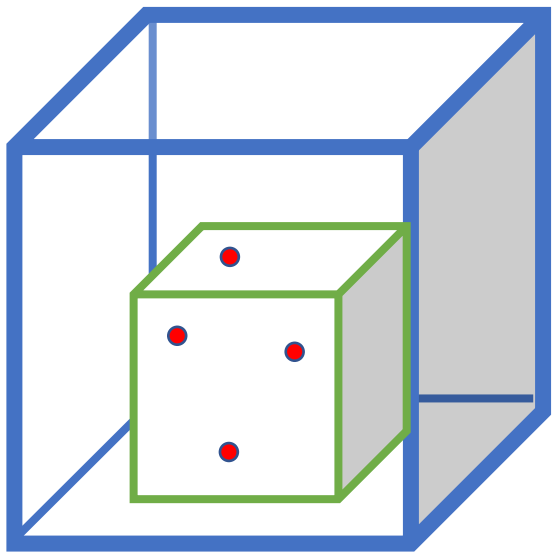


Definition:

- Enc describes a **list-recoverable code** if there are only polynomially many codewords in each product set $S_1 \times S_2 \times \cdots \times S_t$.
- The code is "**algorithmic**" if given $S_1, S_2, \ldots, S_t$, the corresponding messages can be efficiently found.

List-Recoverable Codes

$$\text{Encode: } \{0,1\}^n \to [q]^t$$



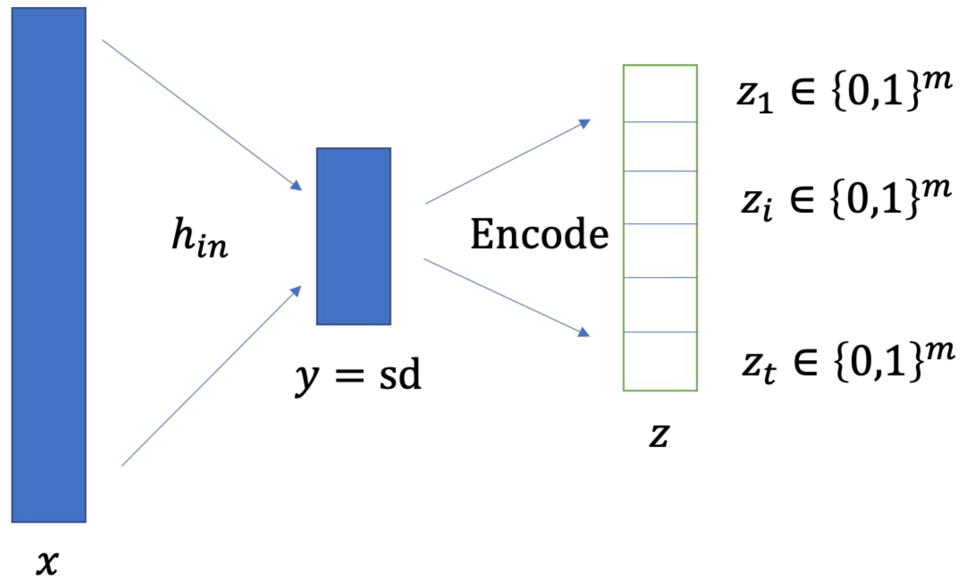
Alternatively: derandomized parallel repetition [BGG90] preserving polynomial number of (efficiently computable) bad challenges

	block-length	number of repetitions (dimension)
	alphabet size	challenge space size for base protocol
		# of bad challenges for base protocol
	"output list" size	# of bad challenge codewords

(t, ℓ, q, L) list-recoverable code

Theorem: Under the **LWE** assumption, there exist CI hash functions for product relations (-> FS for commit-and-open protocols).

Proof Sketch:



Encode is a $(\lambda q, q-1, q)$ list-recoverable code (key lemma)

h_{in} is a [PS19] hash function

Key Lemma: Concatenation of a carefully chosen Parvaresh-Vardy code [PV05] with a poly-size random code has the desired properties.

Extension to Multi-Round Protocols

Theorem: Under the **LWE*** assumption, Fiat-Shamir can be instantiated for any (sufficiently parallel repeated) protocol with:

- Round-by-round soundness [CCHLRRW19], and
- ``efficiently* recognizable bad challenges”

Corollary: FS for parallel repeated Sumcheck or GKR over *small fields* (polynomial or polylogarithmic). [JKKZ20] use exponentially large fields (and don’t need parallel repetition).

Open Problems

- FS for protocols **without efficiently verifiable bad challenges**
  - Graph Isomorphism
  - Commit-and-Open protocols that use Naor/Blum commitments
- Better results for **multi-round protocols**
  - Avoid subexponential assumptions (as in [LV20, JKKZ20, HLR21])
  - Adaptive soundness without leveraging
- Fiat-Shamir for **arguments**? [CJJ21a, **CJJ21b**, LVZ21]
  - **Ingredient**: PCPs with polynomial amount of bad randomness (follows from our codes)

Thank you!