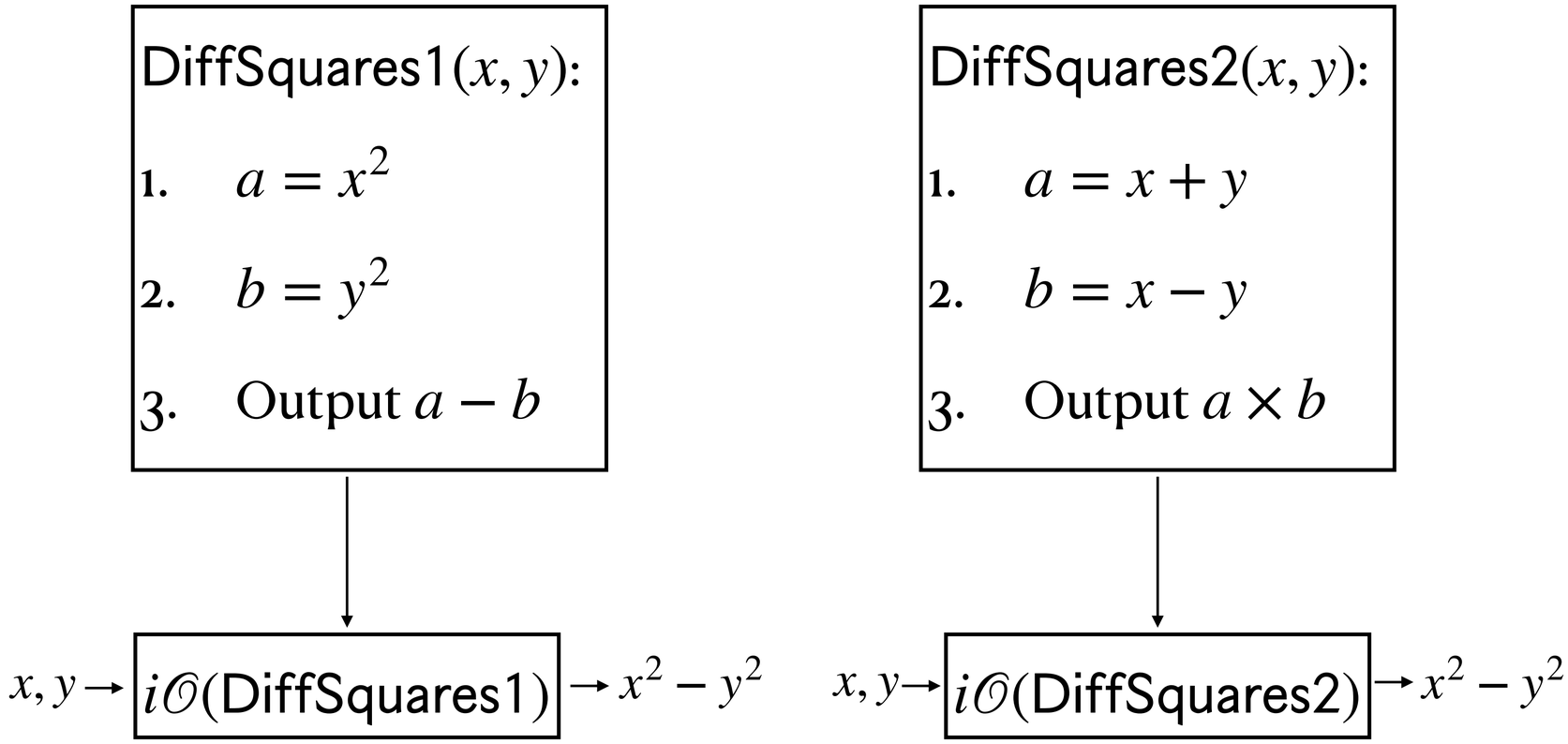
Indistinguishability Obfuscation from Bilinear Maps and LPN Variants

Seyoon Ragavan, Neekon Vafa, Vinod Vaikuntanathan

MIT CSAIL

IO (BGI+01, GGH+13)

From Equivalence to Indistinguishability



IO (BGI+01, GGH+13)

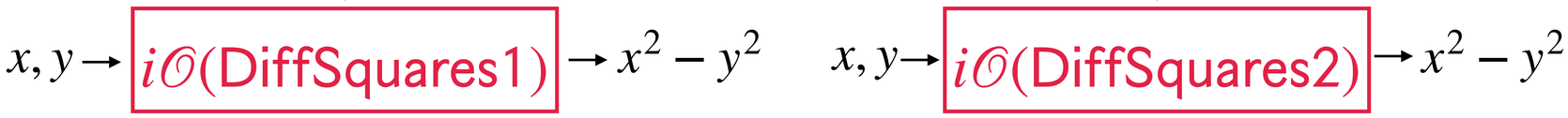
From Equivalence to Indistinguishability

DiffSquares1(x, y):

1. $a = x^2$
2. $b = y^2$
3. Output $a - b$

DiffSquares2(x, y):

1. $a = x + y$
2. $b = x - y$
3. Output $a \times b$



Obfuscated circuits should be computationally indistinguishable

Formal Definition

- PPT obfuscator $i\mathcal{O}$: input and output are circuits from $\{0,1\}^n \to \{0,1\}$
- **Correctness:** for all C and $x \in \{0,1\}^n$, we have $i\mathcal{O}(1^\lambda, C)(x) = C(x)$ w.p. 1
- **Indistinguishability Security:** for all $C_0, C_1 : \{0,1\}^n \to \{0,1\}$ such that:
  - $C_0(x) = C_1(x) \forall x \in \{0,1\}^n$ (functional equivalence), and
  - $|C_0| = |C_1|$ (same size), we have

$$i\mathcal{O}(1^\lambda, C_0) \approx_c i\mathcal{O}(1^\lambda, C_1)\,.$$

(For this talk: $\approx_c$ means poly(λ)-size adversaries can only distinguish with advantage $2^{-\lambda^{\Omega(1)}}$)

IO is "Crypto-Complete"

- Long line of works starting with SW14: *iO* + one-way functions implies essentially all of crypto
  - Fully homomorphic encryption (CLTV15)
  - SNARGs for NP (SW14, WW24)
  - ... and more
- Parallel quest (GGH+13 onwards) to construct *iO*: new constructions, assumptions, and attacks

JLS to the Rescue!

IO from 4 Assumptions

- **Theorem (JLS21):** $i\mathcal{O}$ from sub-exponential security* of the following assumptions:
  - Bilinear maps (SXDH)
  - Learning parity with noise (LPN) over $\mathbb{Z}_p$
  - PRGs in NC^0 from n to $n^{1+\epsilon}$ bits
  - Learning with errors (LWE)



** for a different notion of sub-exponential security*

JLS Strike Again

IO from 3 Assumptions

- **Theorem (JLS22):** $i\mathcal{O}$ from sub-exponential security of the following assumptions:
  - Bilinear maps (DLIN)
  - Learning parity with noise (LPN) over $\mathbb{Z}_p$
  - PRGs in NC^0 from n to $n^{1+\epsilon}$ bits
  - ~~Learning with errors (LWE)~~

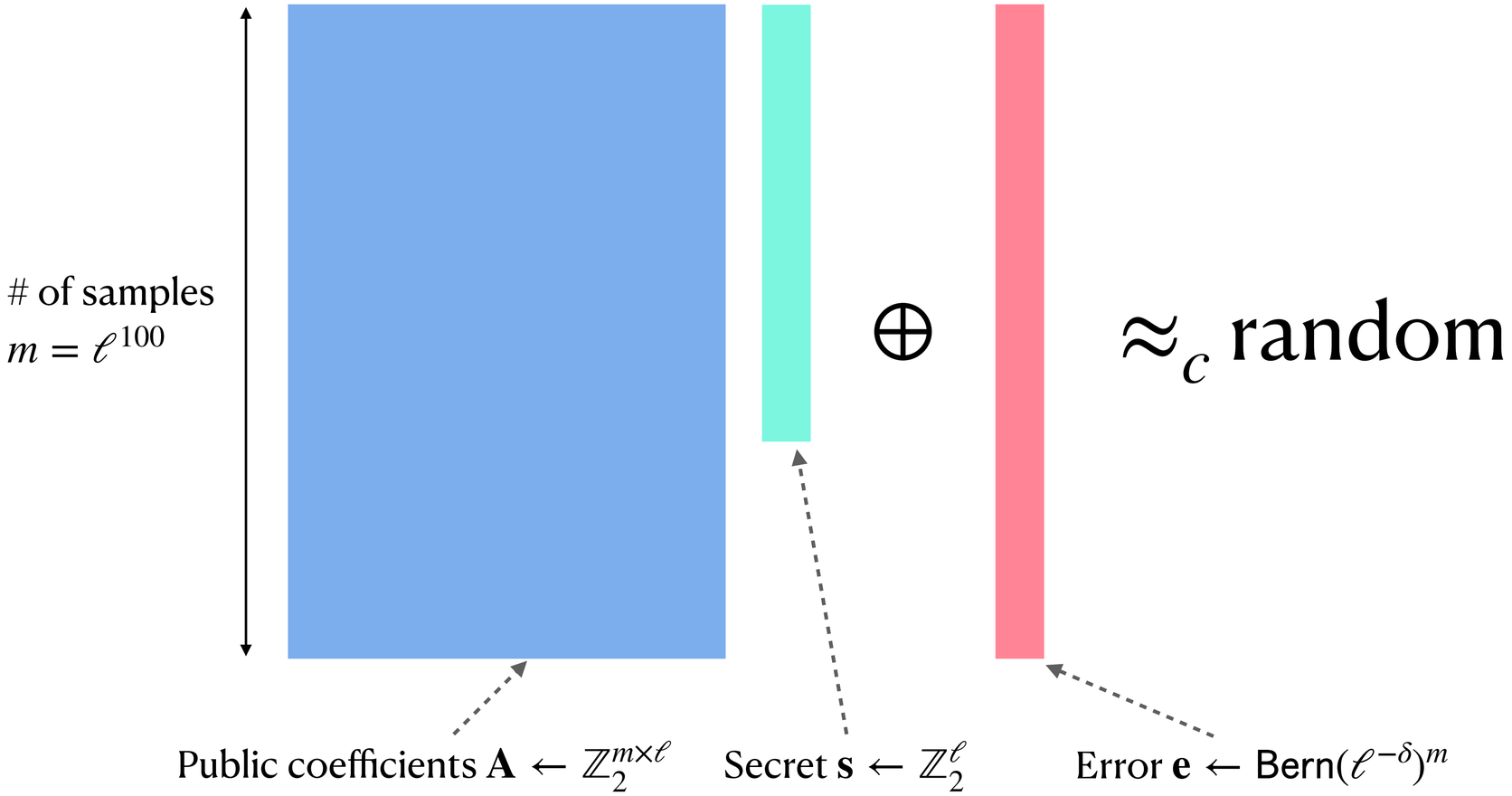


Our Result

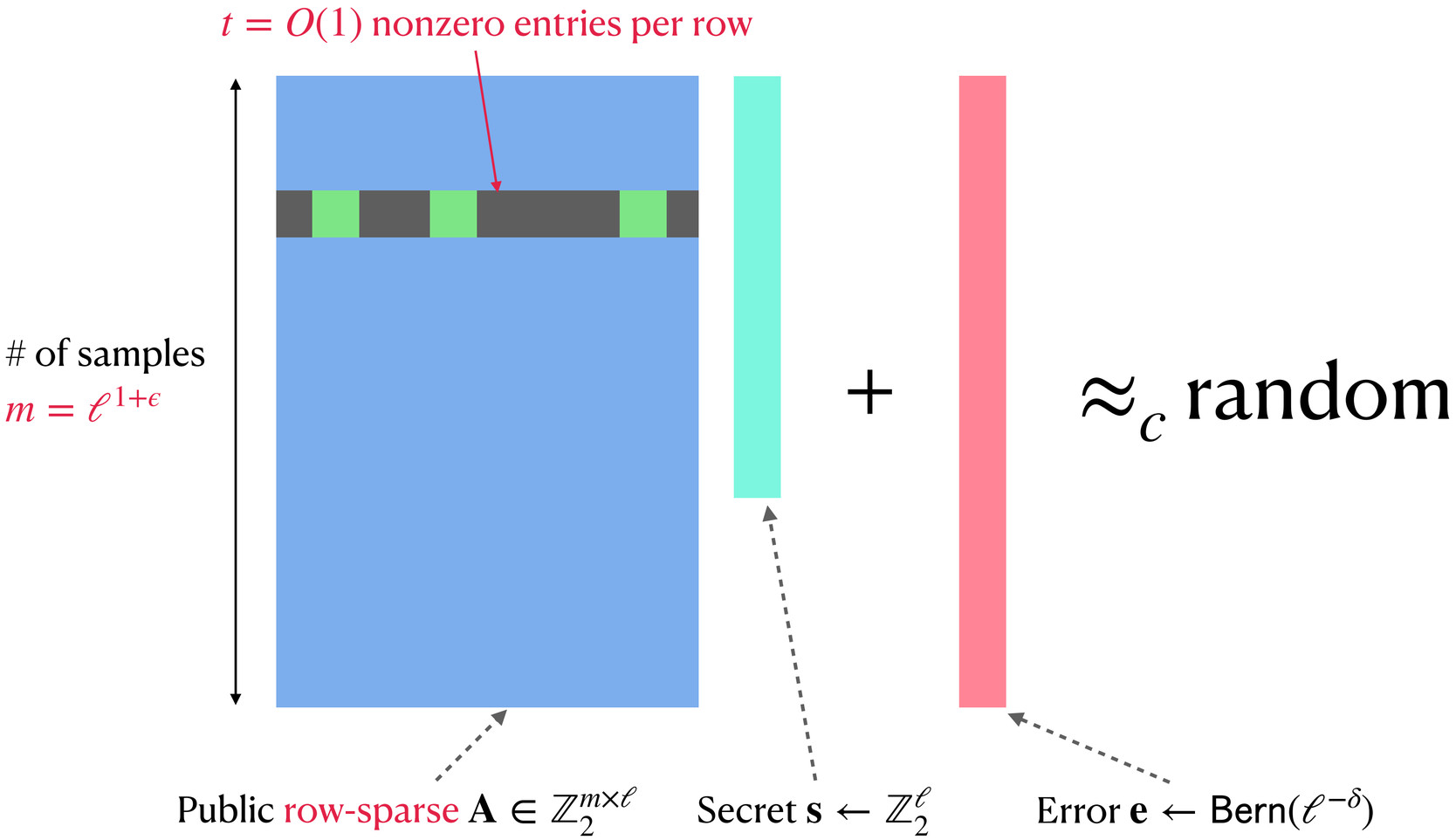
IO from "2.5 Assumptions"

- **Theorem (RVV24):** $i\mathcal{O}$ from sub-exponential security of the following assumptions:
  - Bilinear maps
  - LPN over $\mathbb{Z}_p$
  - ~~PRGs in NC^0 from n to $n^{1+\epsilon}$ bits~~ **Sparse LPN (over $\mathbb{Z}_2$)**
  - ~~LWE~~

LPN (over $\mathbb{Z}_2$)



Sparse LPN* (over $\mathbb{Z}_2$)



* This assumption is kind of... false. 🙃

Let's worry about this later.

Isn't This Easy?

- If Sparse LPN implies PRGs in NC^0, this wouldn't be interesting
- Attempt 1: $G_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) \mapsto \mathbf{As} + \mathbf{e}$
  - $\mathbf{e}$ creates problems: (a) seed is not uniform; (b) $G_{\mathbf{A}}$ is not even expanding
- Attempt 2: $G_{\mathbf{A}}(\mathbf{s}, \mathbf{r}) = \mathbf{As} \oplus \mathsf{Expand}(\mathbf{r})$
  - Correctness: $\mathbf{r}$ uniformly random $\rightarrow \mathsf{Expand}(\mathbf{r})$ distributed like $\mathsf{Bern}(\ell^{-\delta})^m$
  - Locality: Expand should have locality $O(1)$
- Turns out to be impossible: to sample from $\mathsf{Bern}(\eta)$, need locality $\Omega(\log 1/\eta)$
- AIK08: can take $\eta = \Omega(1)$ and obtain NC^0 linear-stretch PRGs from Sparse LPN

Our Observation

- JLS22 needs an NC^0 PRG in two places
  - First use: only need length-doubling: λ bits to 2λ bits
  - Second use: needs slightly polynomial stretch
    - Does not need to work with a uniform seed!
    - Locality is not critical either; just needs to arithmetise as a degree $O(1)$ polynomial over $\mathbb{Z}$

Our Observation

- JLS22's "$\ell^{1+\epsilon}$-stretch PRGs in NC^0" assumption can be replaced by:
  1. Length-doubling PRGs in NC^0; and
  2. SPRG: "structured-seed PRGs with constant $\mathbb{Z}$-degree"
- **Both are implied by Sparse LPN!**
  - (1) follows from AIK08
  - (2) is shown in this work

Generalisation: Local Functions with Noise

Suggested by Aayush Jain, Rachel Lin, anon. TCC reviewer

- Sparse LPN: “local linear function ($\mathbf{As}$) + noise ($\mathbf{e}$)”
  - But we never rely on linearity!

Generalisation: Local Functions with Noise

Suggested by Aayush Jain, Rachel Lin, anon. TCC reviewer

- **LFN:** "local ~~linear~~ function ($f(\mathbf{s})$) + noise ($\mathbf{e}$)"
- Semi-formally:

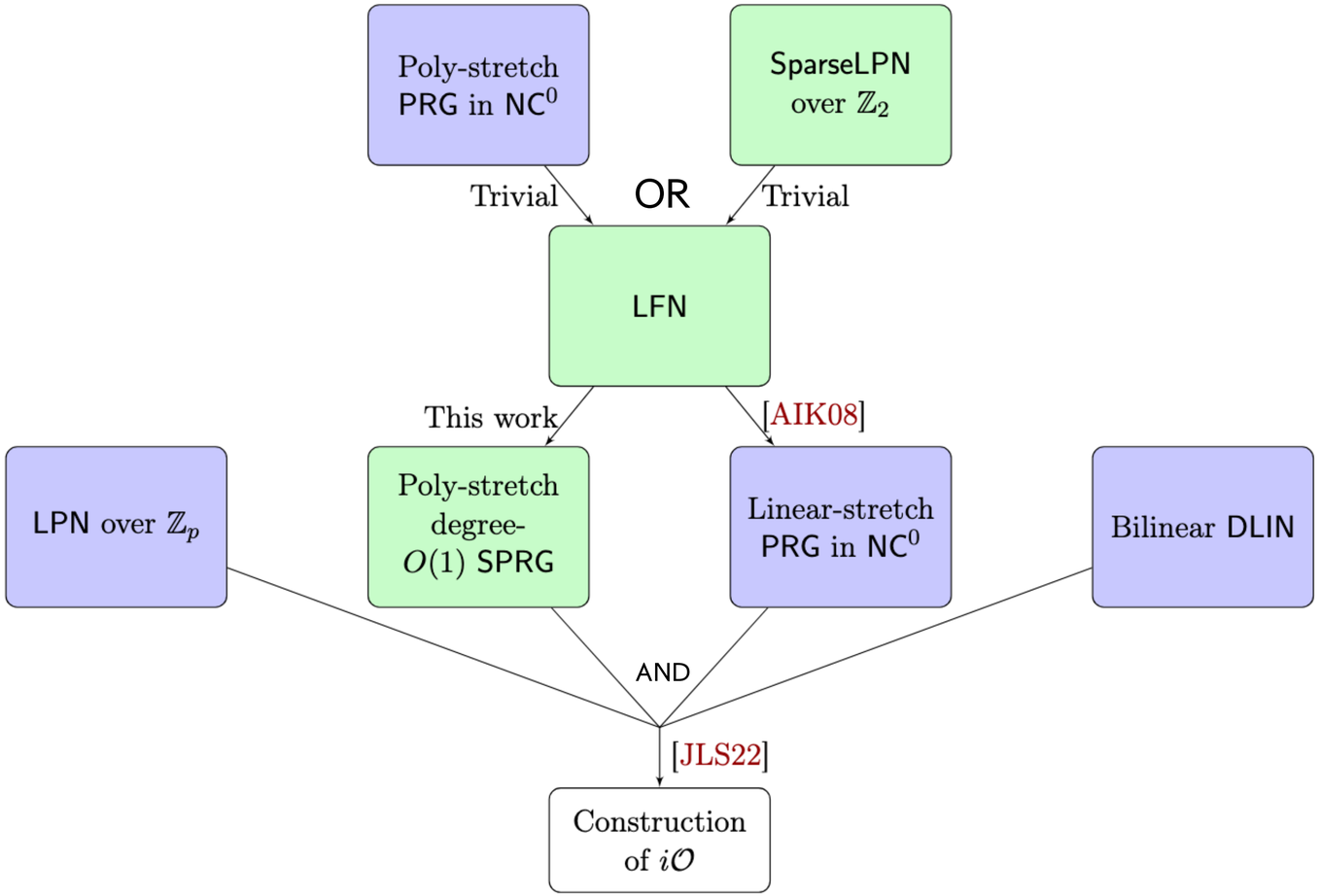
$$(f, f(\mathbf{s}) \oplus \mathbf{e}) \approx_c (f, \mathbf{u})$$

- f is (efficiently) sampled from some distribution on $O(1)$-local functions: $\mathbb{Z}_2^\ell \to \mathbb{Z}_2^m$

How General is LFN?

- Captures both JLS22's and our assumptions:
  - $\mathbf{e} = \mathbf{0}$: PRG in NC^0
    - Not too structured, but prone to algebraic attacks
  - f is linear: Sparse LPN
    - Noise resists algebraic attacks, but has linear structure
- **LFN gets the best of both worlds!**
  - E.g. security of Goldreich's function + 1/poly noise would imply LFN

Our Construction: Summary



Corollaries

- All of the following (and more) from LPN over $\mathbb{Z}_p$, LFN, and DLIN
  - FHE (CLTV15)
  - Perfect ZK adaptively-sound SNARGs and BARGs for NP (WW24, DWW24)
  - Succinct randomised encodings (BGL+15)
  - Witness encryption for NP
  - Multi-party non-interactive key exchange (BZ14, KRS15)
  - Deniable encryption (SW14)
  - Attributed-based encryption
  - Hardness of finding Nash equilibria (AKV04, BPR15, and others)

Notation

- n: input length of the circuit being obfuscated
- $C : \{0,1\}^n \to \{0,1\}$: poly-size circuit to be obfuscated
- Greek letters (δ, ϵ, τ): tiny constant parameters in $(0,1)$
- Latin letters other than d, t: will grow with λ, n
- When describing/analysing algorithms: blue highlight for the obfuscator

Constructing IO: Attempt -1

- $i\mathcal{O}(C) = \mathsf{TT}(C) \in \{0,1\}^{2^n}$: the truth table of C
  - Perfect security
  - WAY too big: $2^n \gg \mathsf{poly}(\lambda, n)$
- Surprisingly, even $2^{n(1-\epsilon)}$ would be sufficient...

XIO (eXponentially efficient IO)

- Three algorithms with runtime poly(λ,2^n):
  - Setup(1^λ,1^{2^n}) $\rightarrow$ crs
  - $\mathcal{X}i\mathcal{O}(1^\lambda, C) \rightarrow \widehat{C}$
  - Eval(1^λ, crs, $\widehat{C}$, $x \in \{0,1\}^n$) $\rightarrow y \in \{0,1\}$
- Functionality: Eval(1^λ, crs, $\widehat{C}$, x) $= C(x)$
- Security: $C_0 \equiv C_1 \Rightarrow$ (crs, $\mathcal{X}i\mathcal{O}(1^\lambda, C_0)$) $\approx_c$ (crs, $\mathcal{X}i\mathcal{O}(1^\lambda, C_1)$)
- Non-trivial compression: $|\ \widehat{C}\ | \leq 2^{n(1-\epsilon)}$

Fast-XIO and Slow-XIO

Definitions

- Slow-XIO: $\mathcal{X}i\mathcal{O}$ can run in time $\mathsf{poly}(\lambda, 2^n)$, even though output length $\leq 2^{n(1-\epsilon)}$
- Fast-XIO: $\mathcal{X}i\mathcal{O}$ needs to run in time $\mathsf{poly}(\lambda, |C|) \cdot 2^{n(1-\epsilon)}$

Bootstrapping Theorems

- **Theorem 1 (AJ15, BV15):** Fast-XIO + OWF implies IO
- **Theorem 2 (GKP+13, LPST16):** Slow-XIO + LWE implies IO*

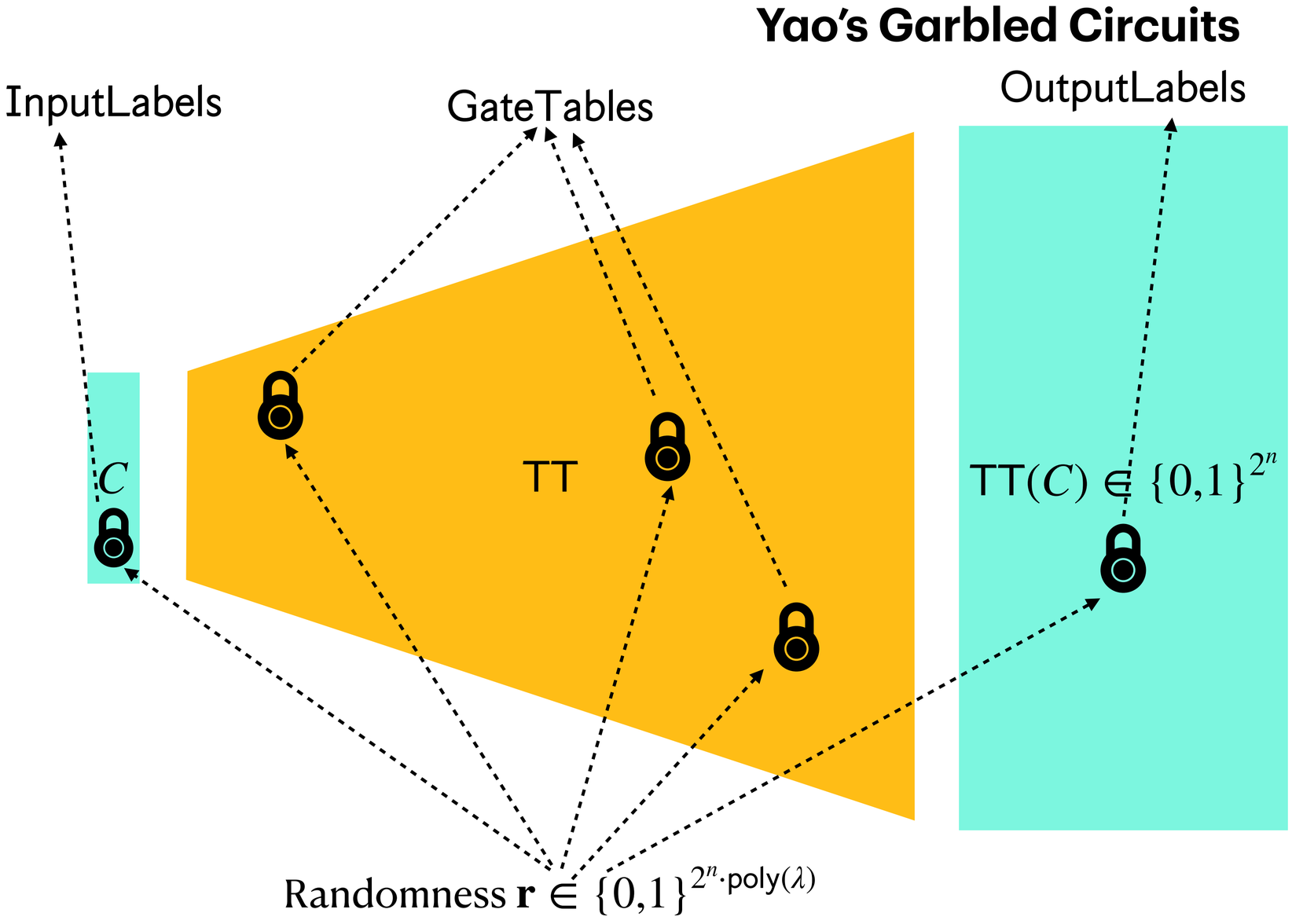
Constructions

- JLS21: complicated construction of Slow-XIO from bilinear SXDH, PRGs in NC^0, LPN over $\mathbb{Z}_p$
- JLS22: cleaner construction of Fast-XIO from the same assumptions (DLIN instead of SXDH)
- RVV24: modify JLS22 to obtain Fast-XIO from the same assumptions but with Sparse LPN instead of NC^0 PRGs

** LPST16 contains a bug; true with some modifications*

JLS22: An Overview

Starting Point: Randomised Encodings



- Instantiate Yao with a **local** length-doubling PRG
- InputLabels, GateTables, OutputLabels all local functions of C, $\mathbf{r}$
- All outputs together: $\mathsf{RE}(1^\lambda, \mathsf{TT}, (C||\mathbf{r}))$

Constructing XIO: Attempt 0

(terrible, but bear with me)

- $\mathcal{X}i\mathcal{O}(1^\lambda, C)$: $\widehat{C} = (C, \mathbf{r})$
- $\mathsf{Eval}(1^\lambda, \widehat{C}, x \in \{0,1\}^n)$:
  - Compute $(C, \mathbf{r}) \mapsto (\mathsf{InputLabels}, \mathsf{GateTables}, \mathsf{OutputLabels})$
  - Yao's garbled circuit evaluation $\rightarrow$ recover $\mathsf{TT}(C) \rightarrow C(x)$

Two problems

1. $|\mathbf{r}| = 2^n \cdot \mathsf{poly}(\lambda) \rightarrow$ no compression
2. Sending $C, \mathbf{r}$ in the clear $\rightarrow$ completely insecure

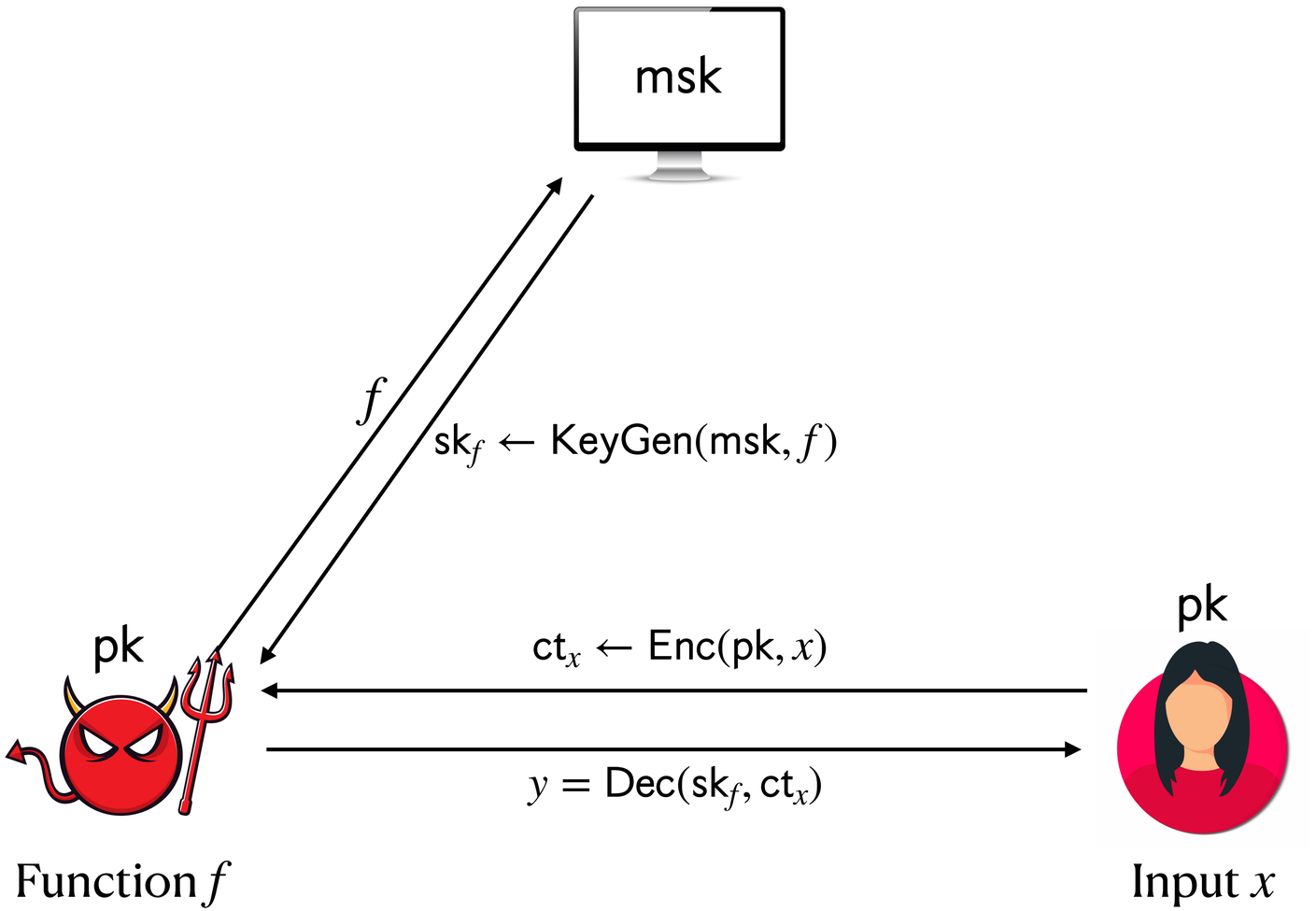
Problem 1: Compression

- Need $2^n \cdot \mathsf{poly}(\lambda)$ random bits
- **Solution:** make these bits pseudorandom
  - $\mathcal{X}i\mathcal{O}$: sample random $\mathbf{r}' \leftarrow \{0,1\}^{2^{n(1-\epsilon)} \cdot \mathsf{poly}(\lambda)}$ and output $(C, \mathbf{r}')$
  - Eval: compute $\mathsf{RE}(1^\lambda, \mathsf{TT}, (C \,||\, G(\mathbf{r}'))$, then evaluate as in Yao
- Security still a problem
- But if G is local **(poly-stretch PRG in** NC^0**)**, at least Eval is local in the outputs of $\mathcal{X}i\mathcal{O}$

Problem 2: Security

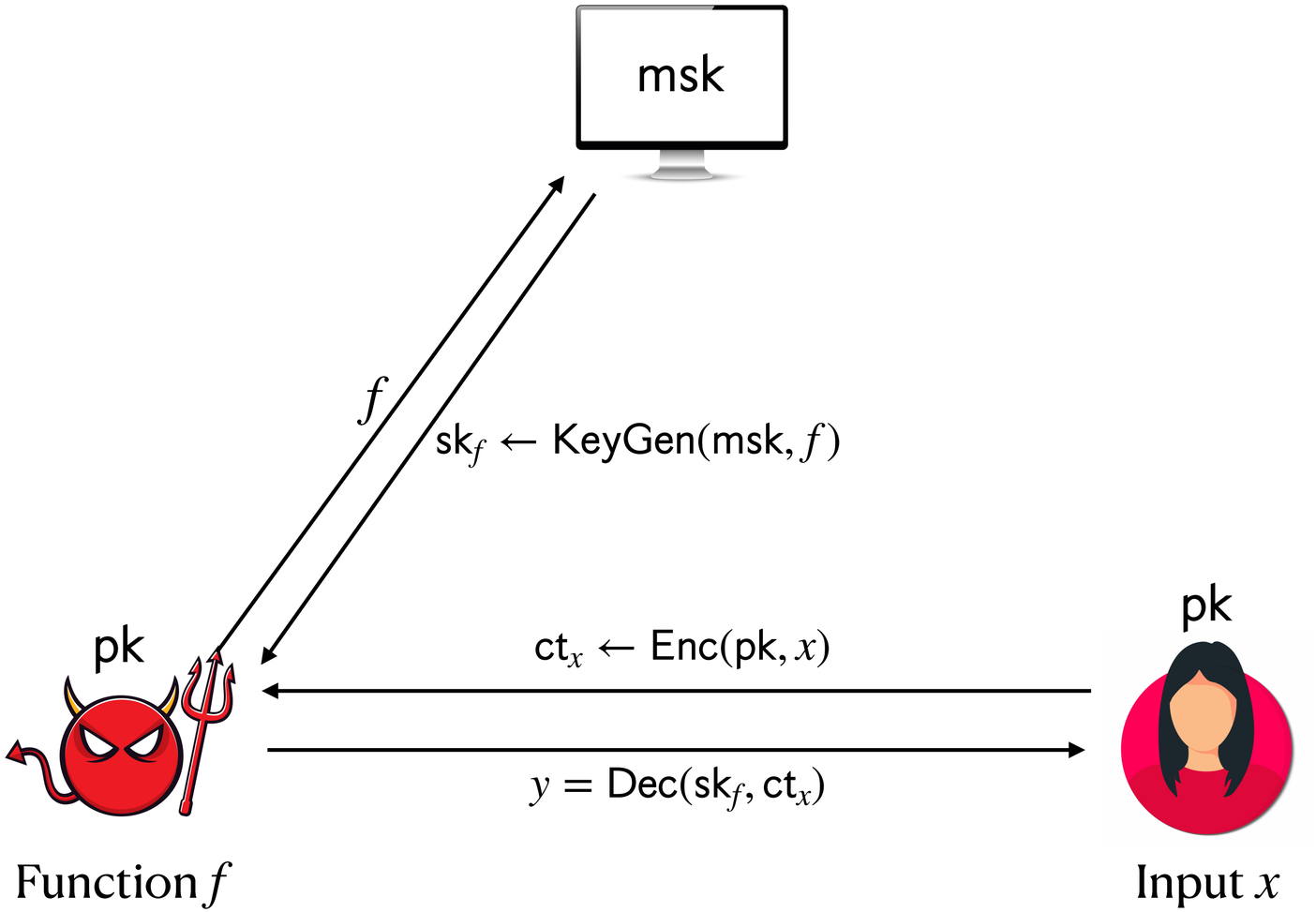
- Need $\mathcal{X}i\mathcal{O}$ to hide $C, \mathbf{r}'$ in a way that still allows Eval to compute a local function of them
- Observation: local Boolean functions are degree $O(1)$ over $\mathbb{Z}$
  - E.g. $x \vee y = x + y - xy; x \wedge y = xy$
- Tool (morally): functional encryption for degree $O(1)$ polynomials

Functional Encryption



- Correctness: $y = f(x)$
- Security: Eve doesn't learn anything about x beyond $f(x)$

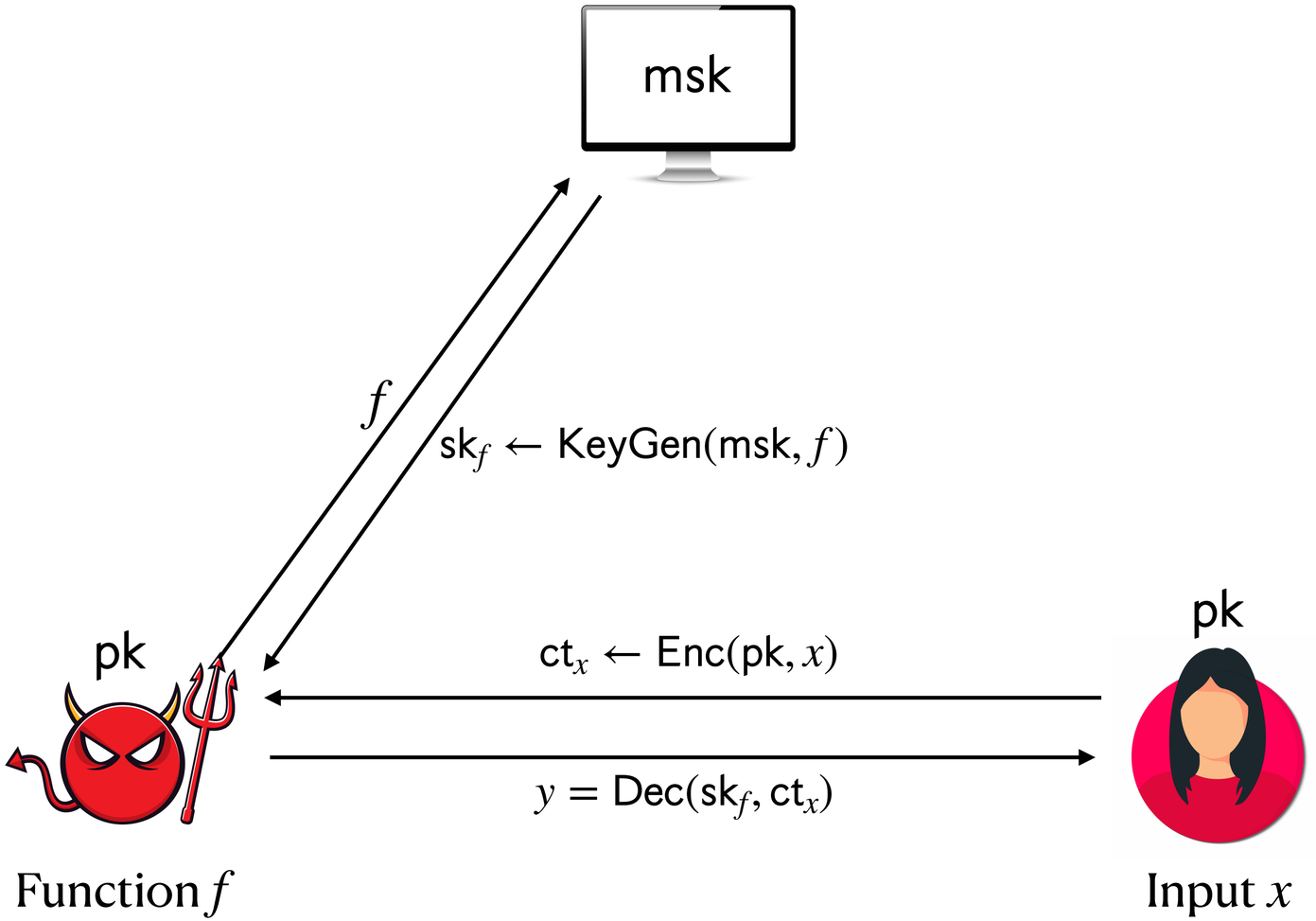
From FE (for Degree $O(1)$) to XIO



Insecure XIO from before:

- $\mathcal{XiO}$: output $(C, \mathbf{r}')$
- Eval: compute $\mathsf{RE}(1^\lambda, \mathsf{TT}, (C, G(\mathbf{r}')))$ then use Yao's garbled circuit evaluation to obtain $\mathsf{TT}(C)$

From FE (for Degree $O(1)$) to XIO



Secure XIO: throw this under the FE hood!

- $\mathcal{XIO}$: output $\widehat{C} = \mathsf{Enc}(\mathsf{pk}, (C, \mathbf{r}'))$
- Setup: compute $\mathsf{crs} = \mathsf{KeyGen}(\mathsf{msk}, \mathsf{RE}(1^\lambda, \mathsf{TT}, (\cdot, G(\cdot))))$
- Eval: compute $\mathsf{Dec}(\mathsf{crs}, \widehat{C})$ then use Yao to obtain $\mathsf{TT}(C)$

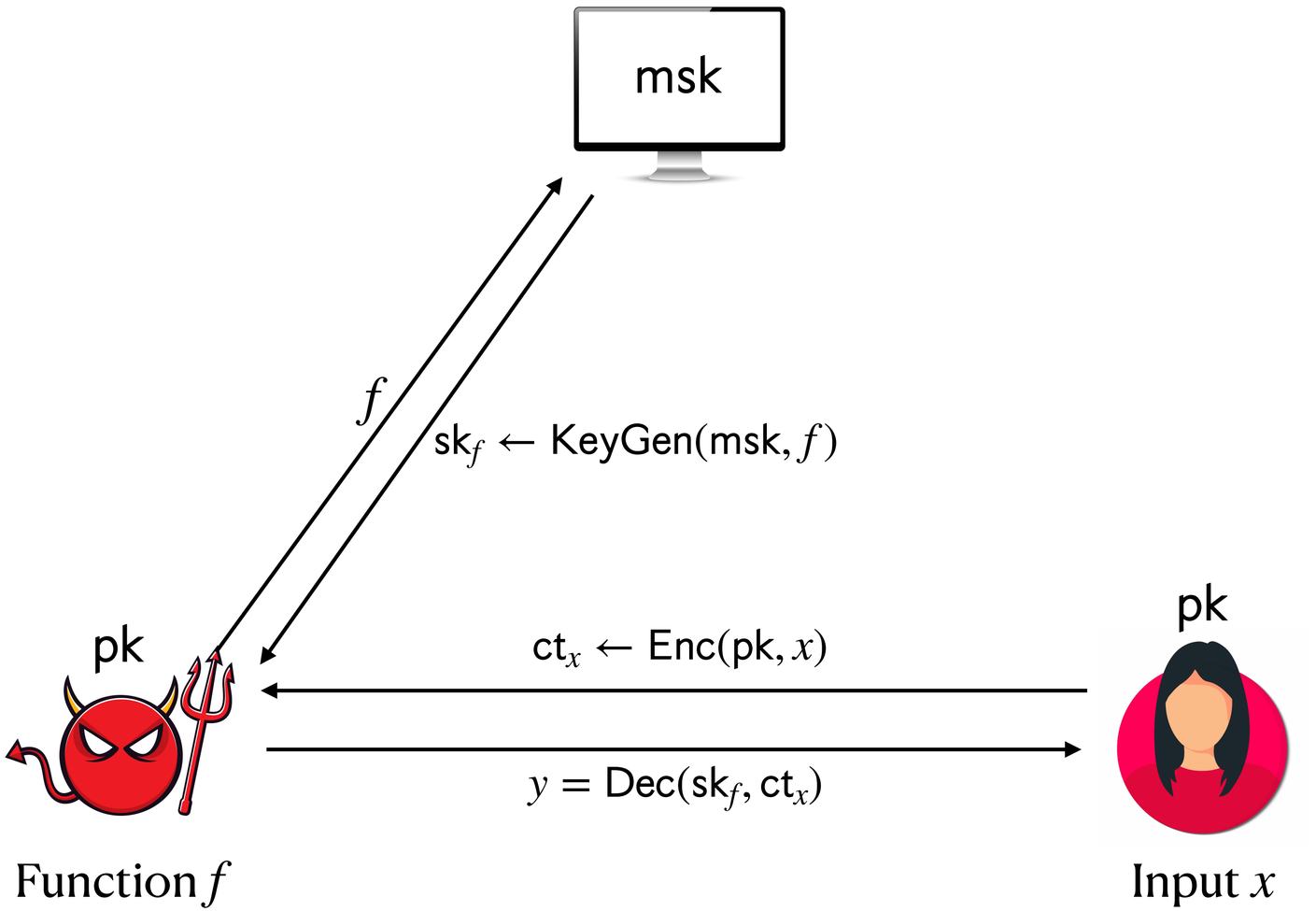
Constructing Degree $O(1)$ FE

(technical caveats; do not try this at home)

- Initial attempts at constructing IO (GGH+13, CLT13): instantiate degree $O(1)$ FE using multilinear maps
  - All broken by 2016; no known instantiations

1. **Theorem (Wee20):** bilinear DLIN implies FE for degree 2
2. **"Theorem" (JLS22):** LPN over $\mathbb{Z}_p$ can be used to boost degree 2 FE to degree $O(1)$ FE

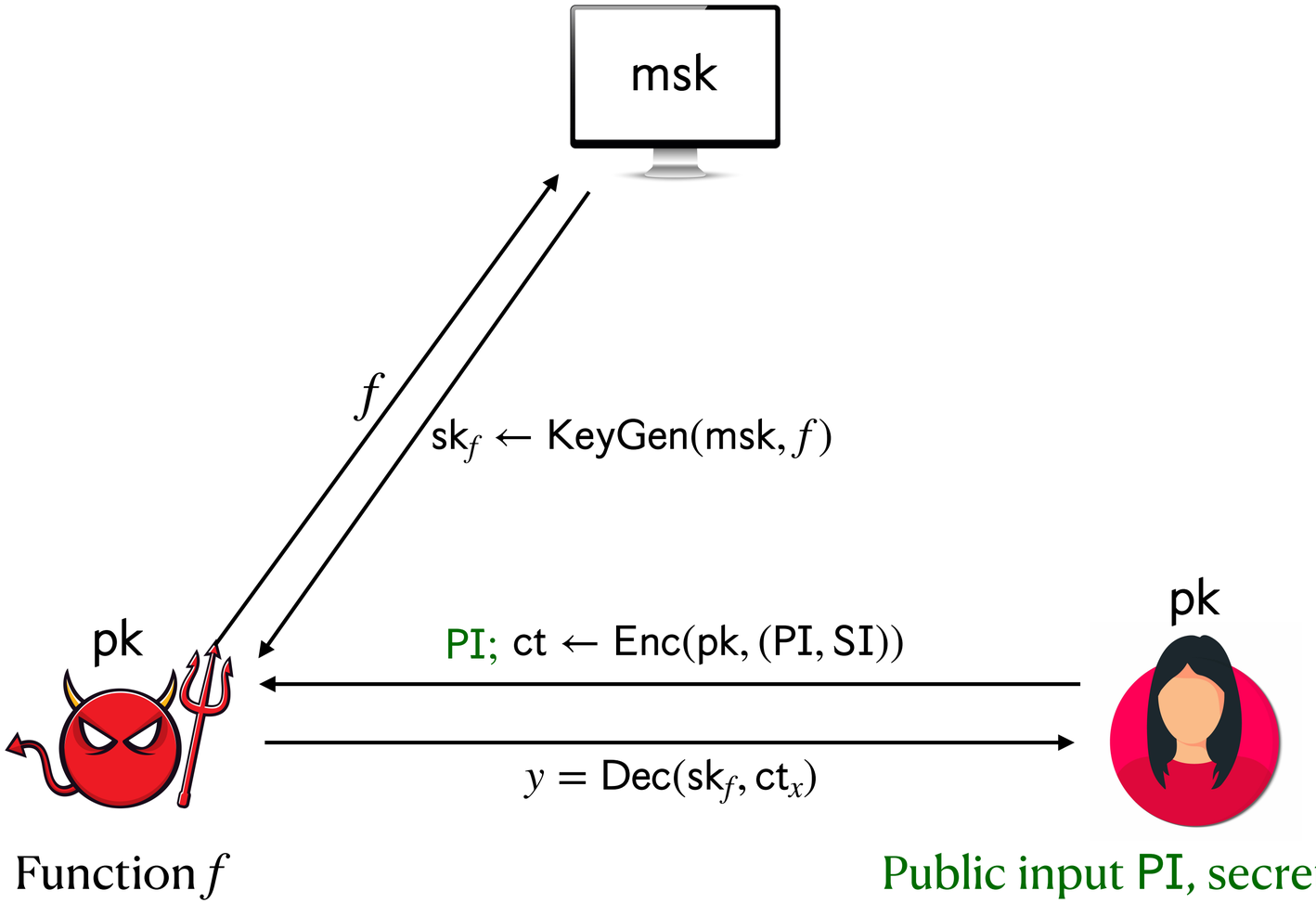
Reminder: Functional Encryption



- Correctness: $y = f(x)$
- Security: Eve doesn't learn anything about x beyond $f(x)$

Tool: Partially Hiding Functional Encryption

(Hides degree $(O(1), 2)$ computations)



- Correctness: $y = f(\mathsf{PI}, \mathsf{SI})$, provided $f(\mathsf{PI}, \mathsf{SI}) \in \{0,1\}$
- Security: Eve doesn't learn anything about SI beyond $f(\mathsf{PI}, \mathsf{SI})$
- Wee20: instantiates from bilinear groups if f is degree $O(1)$ in PI and 2 in SI
- Efficiency: runtime of Enc is $(|\mathsf{PI}| + |\mathsf{SI}|) \cdot \mathsf{poly}(\lambda)$

Boosting to Degree $O(1)$ FE

Regev-Style Homomorphic Encryption

- $\mathsf{HEEnc}(C||\mathbf{r})$: $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} + (C||\mathbf{r}) \pmod{p}$
  - $\mathbf{s} \leftarrow \mathbb{Z}_p^{\ell}, \mathbf{A} \leftarrow \mathbb{Z}_p^{m \times \ell}$
  - Parameters: $m \approx 2^{n(1-\epsilon)}, \ell \approx m^{\tau}$
- Typical linear HE decryption: evaluate $f(\mathbf{b} - \mathbf{A}\mathbf{s})$ then round to handle errors from $\mathbf{e}$
- Two problems:
  1. Can't give out $\mathbf{s}$
  2. f is nonlinear (degree $d = O(1)$) $\rightarrow$ may not even get correctness

Problem 1: Hiding s

- Solution: shove $\mathbf{s}$ into the SI of the PHFE!
  - PI: $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e} + (C||\mathbf{r}))$
  - SI: $(1||\mathbf{s})^{\otimes d/2}$
- Now $f(\mathbf{b} - \mathbf{As})$ is degree $(d,\ 2)$ in $(\mathsf{PI}, \mathsf{SI})$
- Need to ensure $|\mathsf{PI}| + |\mathsf{SI}| \leq 2^{n(1-\epsilon)}$:
  - $|\mathsf{PI}|: m \cdot \ell \approx 2^{n(1-\epsilon)} \cdot 2^{n(1-\epsilon)\tau} \leq 2^{n(1-\epsilon')}$ for small enough τ
  - $|\mathsf{SI}|: (\ell + 1)^{d/2} \approx m^{d\tau/2} \leq 2^{n(1-\epsilon')}$ for small enough τ

Reminder: $|\mathsf{PI}| + |\mathsf{SI}|$ → PHFE ciphertext size $= |\mathscr{X}i\mathcal{O}(1^\lambda, C)|$

Problem 2: Correctness

- We can now securely compute $f(\mathbf{b} - \mathbf{As}) = f((C||\mathbf{r}) + \mathbf{e})$
- So remains to compute

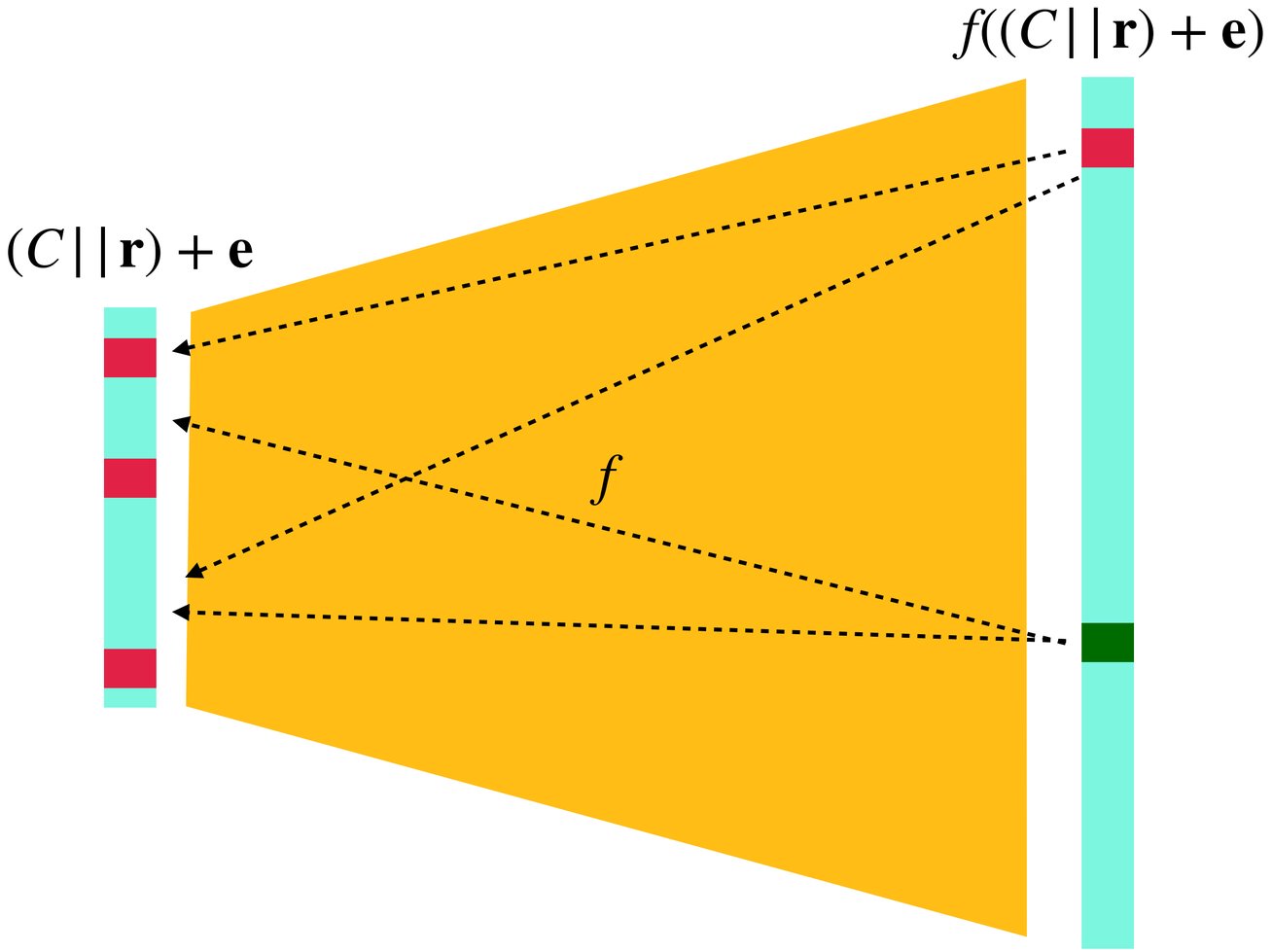
$$\mathsf{err} := f(C||\mathbf{r}) - f((C||\mathbf{r}) + \mathbf{e})$$

  Then we can recover $f(\mathbf{b} - \mathbf{As}) + \mathsf{err} = \mathsf{RE}(1^\lambda, \mathsf{TT}; (C, G(\mathbf{r})))$.

- If $\mathbf{e}$ is small (LWE): hard to say much about this

The JLS idea: if $\mathbf{e}$ is sparse (as in LPN), then err is also sparse!

Why is err Sparse?



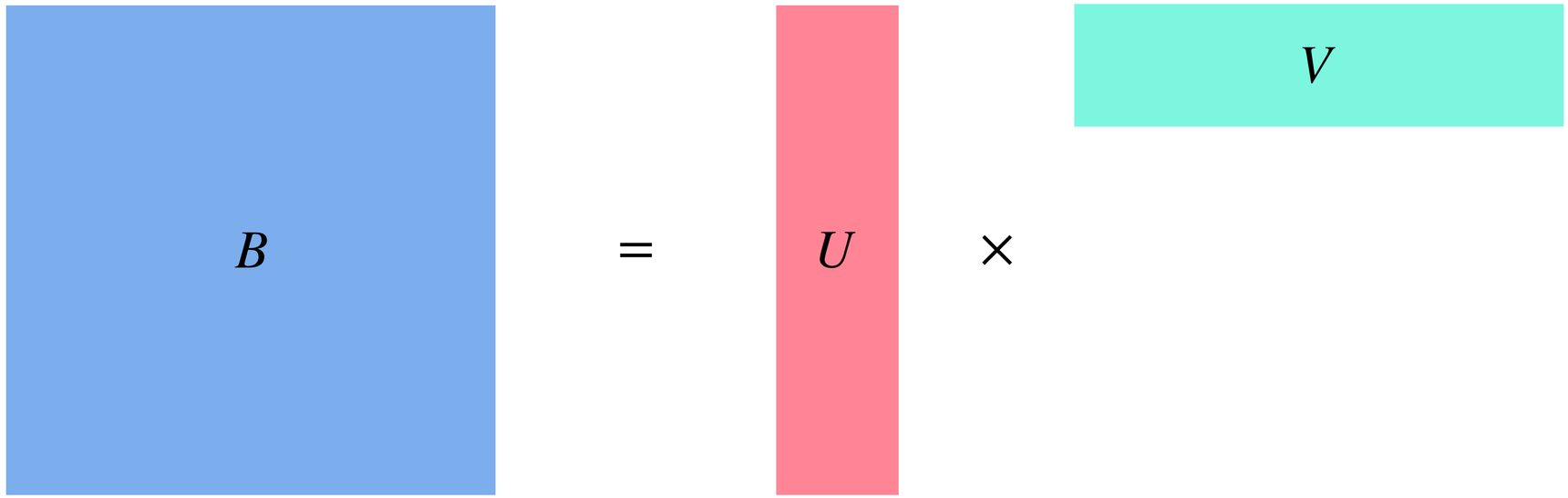
- Assume $\mathbf{e} \in \mathbb{Z}_p^m$ has $m^{1-\delta}$ nonzero entries
- f has locality $O(1) \rightarrow$ probability of each output entry being corrupt is also $O(m^{-\delta})$
- Expected number of corrupt entries: $O(2^n \cdot m^{-\delta}) = 2^{n(1-\Omega(1))}$.

Solving Problem 2: Compress err

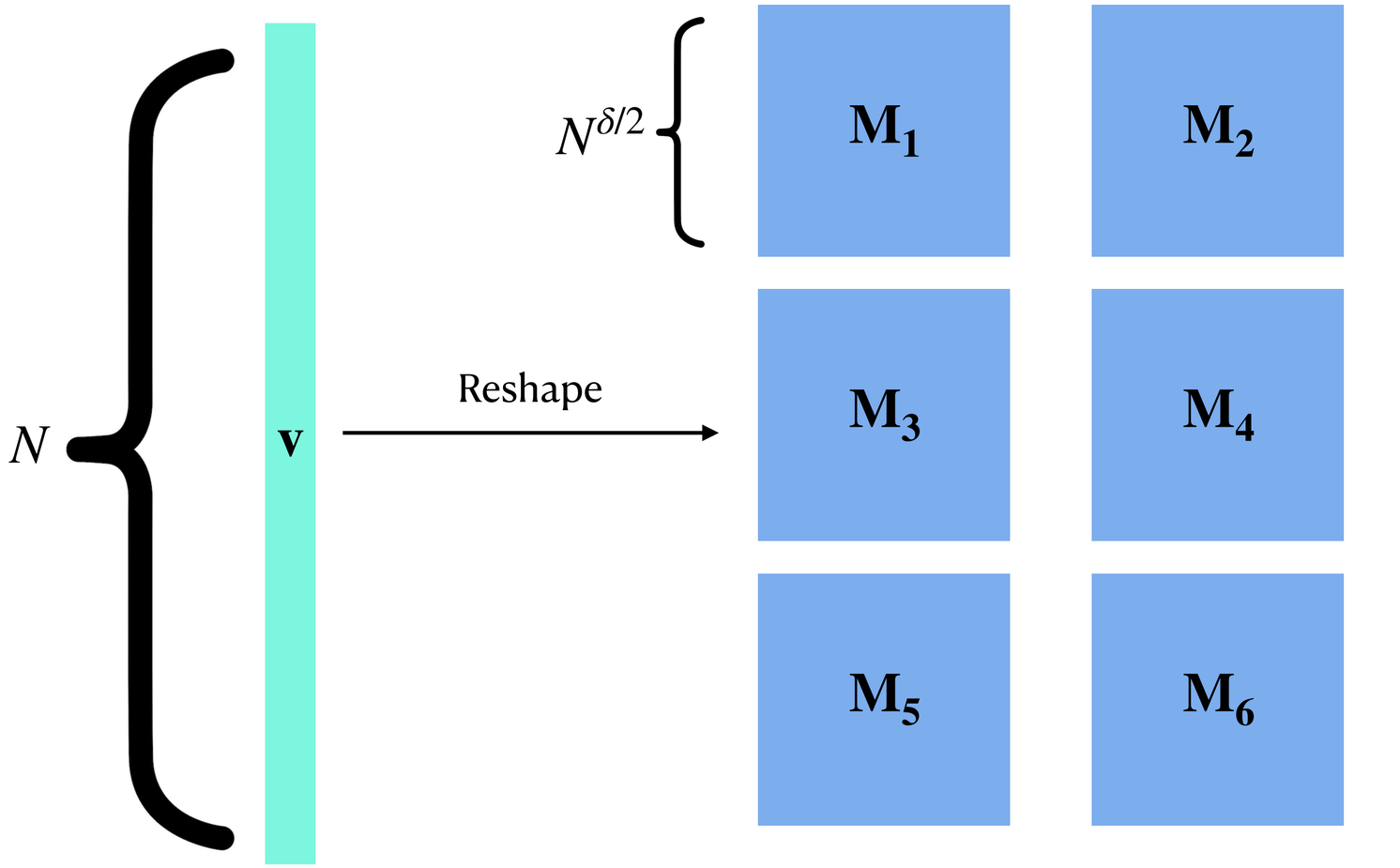
- JLS show that sparse vectors in $\mathbb{Z}_p^N$ can be compressed such that:
  - Compress($\mathbf{v}$) outputs a vector $\hat{\mathbf{v}}$ of length $N^{1-\Omega(1)}$; and
  - Expand : $\hat{\mathbf{v}} \mapsto \mathbf{v}$ is a degree 2 operation
- If we do this, we can just include $\hat{\mathbf{v}}$ in SI!

The JLS Compression Trick

- Observation 1: $M \times M$ **matrices** with sparsity $M^{0.9}$ also have rank $\leq M^{0.9}$
- Observation 2: low-rank matrices can be decomposed as a product of two small matrices: **degree 2 in** (U, V)**!**



The JLS Compression Trick

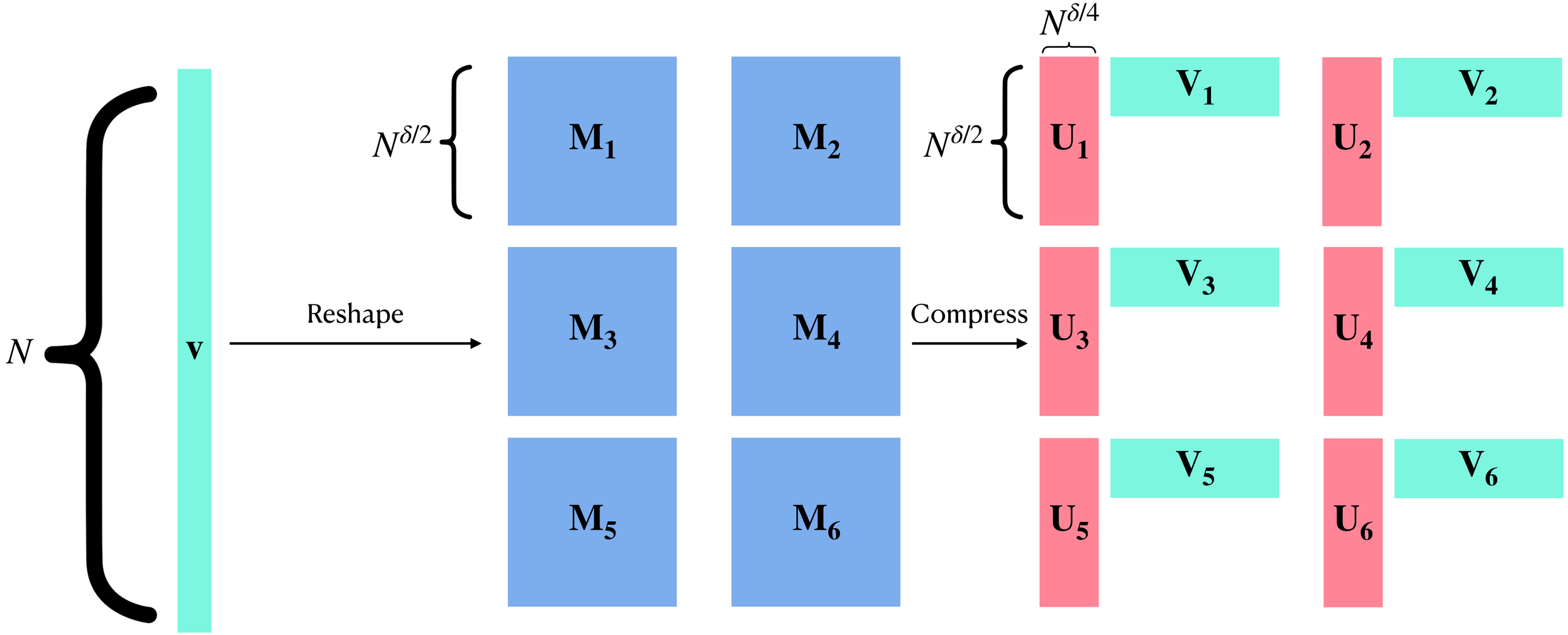


Length N vector with sparsity $N^{1-\delta}$

$N^{1-\delta}$ matrices of dimensions $N^{\delta/2} \times N^{\delta/2}$

- Each $\mathbf{M_i}$ has N^{δ} entries
  - Expected # of nonzero entries is $O(1)$
  - Whp, # of nonzero entries is $\leq N^{\delta/4}$
- So we can compress each of these!

The JLS Compression Trick



Length N vector with sparsity $N^{1-\delta}$

$N^{1-\delta}$ matrices of dimensions $N^{\delta/2} \times N^{\delta/2}$

Total size: $O(N^{1-\delta} \times N^{\delta/2} \times N^{\delta/4}) = O(N^{1-\delta/4})$

JLS22: Summary

(For getting to Slow-XIO)

- Setup$(1^\lambda, 1^{2^n})$:
  - (pk, msk) $\leftarrow$ PHFE . Setup
  - $\mathsf{sk}_f \leftarrow$ PHFE . KeyGen(msk, f)
  - Output crs $=$ (pk, sk_f)
- Eval$(1^\lambda, \mathsf{crs}, \widehat{C}, x)$:
  - Use PHFE . Dec(sk_f, $\cdot$) to recover
    $f(\mathsf{PI}, \mathsf{SI}) := \mathsf{RE}(1^\lambda, \mathsf{TT}, \mathbf{b} - \mathbf{As}) + \mathsf{Reshape}\left(\{\mathbf{U_i} \cdot \mathbf{V_i}\}\right)$
    $= \mathsf{RE}(1^\lambda, \mathsf{TT}, (C||\mathbf{r}))$
  - Yao $\rightarrow \mathsf{TT}(C) \rightarrow C(x)$

- $\mathcal{XIO}(1^\lambda, C)$:
  - Sample $\mathbf{r} \leftarrow \{0,1\}^{2^{n(1-\epsilon)}}$
  - Let
    $\mathsf{PI} = (\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e} + (C||\mathbf{r}))$
  - Let $\mathsf{SI} = ((1||\mathbf{s})^{\otimes d/2}, \{\mathbf{U_i}, \mathbf{V_i}\})$
  - Output
    $\widehat{C} =$ PHFE . Enc(pk, (PI, SI))

Our Modification with Sparse LPN

Where did we need NC^0 PRGs?

- Two points:
  1. Length-doubling PRG inside Yao's garbled circuits
  2. Expanding $2^{n(1-\epsilon)}$ random bits to 2^n pseudorandom bits
- **Our observation: for application (2), the seed doesn't need to be random or even pseudorandom!**
- *For application (1), the seed actually has to be (pseudo)random*

Structured-Seed PRGs (SPRGs)

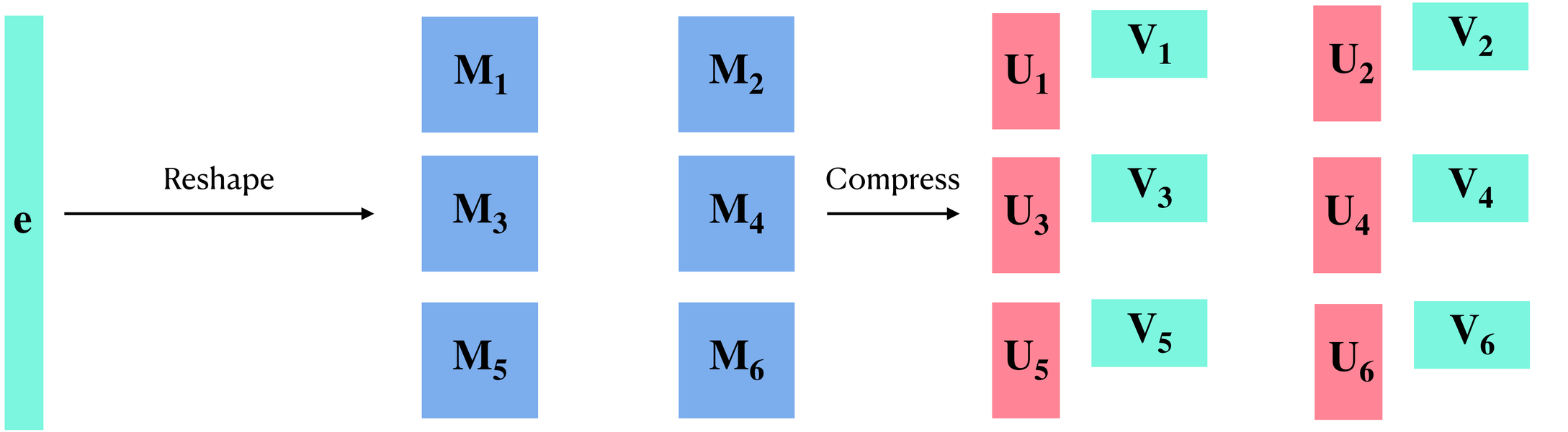
- Three algorithms:
  - $\mathsf{IdSamp}(1^{2^n})$: outputs a function index I
  - $\mathsf{SdSamp}(2^n)$: outputs seed $\in \{0,1\}^{2^{n(1-\epsilon)}}$
  - $\mathsf{Eval}(I, \mathsf{seed})$: outputs randomness $\mathbf{r} \in \{0,1\}^{2^n}$
- Properties:
  - Polynomial efficiency: $\mathsf{Eval}(I, \cdot)$ has degree $O(1)$ and locality $2^{n\delta}$ over $\mathbb{Z}$
  - Security: Eval's output is pseudorandom
  - Sublinear efficiency (for Fast-XIO): SdSamp runs in time $2^{n(1-\epsilon')}$

Generalising JLS22

- **Theorem (essentially JLS22):** Assuming all of the following, we can get IO:
  - Bilinear DLIN
  - LPN over $\mathbb{Z}_p$
  - Linear-stretch PRG in NC^0
  - Structured-seed PRG
- **Theorem (AIK08):** Assuming Sparse LPN, there exists a linear-stretch PRG in NC^0
- **Theorem (this work):** Assuming Sparse LPN, there exists a structured-seed PRG

JLS Compression Returns!

- Issue with directly instantiating PRGs in NC^0 from Sparse LPN: compressing the error to a uniform seed
- But JLS already gives us a way to compress it to a structured seed!



Length N vector with sparsity $N^{1-\delta}$

$N^{1-\delta}$ matrices of dimensions $N^{\delta/2} \times N^{\delta/2}$

Total size: $O(N^{1-\delta} \times N^{\delta/2} \times N^{\delta/4}) = O(N^{1-\delta/4})$

Our SPRG Construction from Sparse LPN

- IdSamp(1^{2^n}):
  - Sample and output row-sparse $\mathbf{A} \in \{0,1\}^{2^n \times 2^{n(1-\epsilon)}}$
- SdSamp(2^n):
  - Sample $\mathbf{s} \leftarrow \{0,1\}^{2^{n(1-\epsilon)}}$
  - Sample sparse $\mathbf{e} \in \{0,1\}^{2^n}$, compress to $\{\mathbf{U_i}, \mathbf{V_i}\}$
  - Output $\left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)$
- Eval $\left(\mathbf{A}, \left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)\right)$
  - Compute all $\mathbf{U_i} \cdot \mathbf{V_i}$ to obtain $\mathbf{e}$
  - Output $\mathbf{As} \oplus \mathbf{e}$

Our SPRG Construction from Sparse LPN

- IdSamp(1^{2^n}):
  - Sample and output row-sparse $\mathbf{A} \in \{0,1\}^{2^n \times 2^{n(1-\epsilon)}}$
- SdSamp(2^n):
  - Sample $\mathbf{s} \leftarrow \{0,1\}^{2^{n(1-\epsilon)}}$
  - Sample sparse $\mathbf{e} \in \{0,1\}^{2^n}$, compress to $\{\mathbf{U_i}, \mathbf{V_i}\}$
  - Output $\left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)$
- Eval $\left(\mathbf{A}, \left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)\right)$
  - Compute all $\mathbf{U_i} \cdot \mathbf{V_i}$ to obtain $\mathbf{e}$
  - Output $\mathbf{As} \oplus \mathbf{e}$

Polynomial Efficiency:

- Step 1: degree 2, locality $O(2^{n\delta/4})$

Our SPRG Construction from Sparse LPN

- IdSamp(1^{2^n}):
  - Sample and output row-sparse $\mathbf{A} \in \{0,1\}^{2^n \times 2^{n(1-\epsilon)}}$
- SdSamp(2^n):
  - Sample $\mathbf{s} \leftarrow \{0,1\}^{2^{n(1-\epsilon)}}$
  - Sample sparse $\mathbf{e} \in \{0,1\}^{2^n}$, compress to $\{\mathbf{U_i}, \mathbf{V_i}\}$
  - Output $\left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)$
- Eval $\left(\mathbf{A}, \left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)\right)$
  - Compute all $\mathbf{U_i} \cdot \mathbf{V_i}$ to obtain $\mathbf{e}$
  - Output $\mathbf{As} \oplus \mathbf{e}$

Polynomial Efficiency:

- Step 1: degree 2, locality $O(2^{n\delta/4})$
- Step 2: locality $O(t)$ (sparsity parameter) → degree $O(t)$ too

Our SPRG Construction from Sparse LPN

- $\mathsf{IdSamp}(1^{2^n})$:
  - Sample and output row-sparse $\mathbf{A} \in \{0,1\}^{2^n \times 2^{n(1-\epsilon)}}$
- $\mathsf{SdSamp}(2^n)$:
  - Sample $\mathbf{s} \leftarrow \{0,1\}^{2^{n(1-\epsilon)}}$
  - Sample sparse $\mathbf{e} \in \{0,1\}^{2^n}$, compress to $\{\mathbf{U_i}, \mathbf{V_i}\}$
  - Output $\left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)$
- $\mathsf{Eval}\left(\mathbf{A}, \left(\mathbf{s}, \{\mathbf{U_i}, \mathbf{V_i}\}\right)\right)$
  - Compute all $\mathbf{U_i} \cdot \mathbf{V_i}$ to obtain $\mathbf{e}$
  - Output $\mathbf{As} \oplus \mathbf{e}$

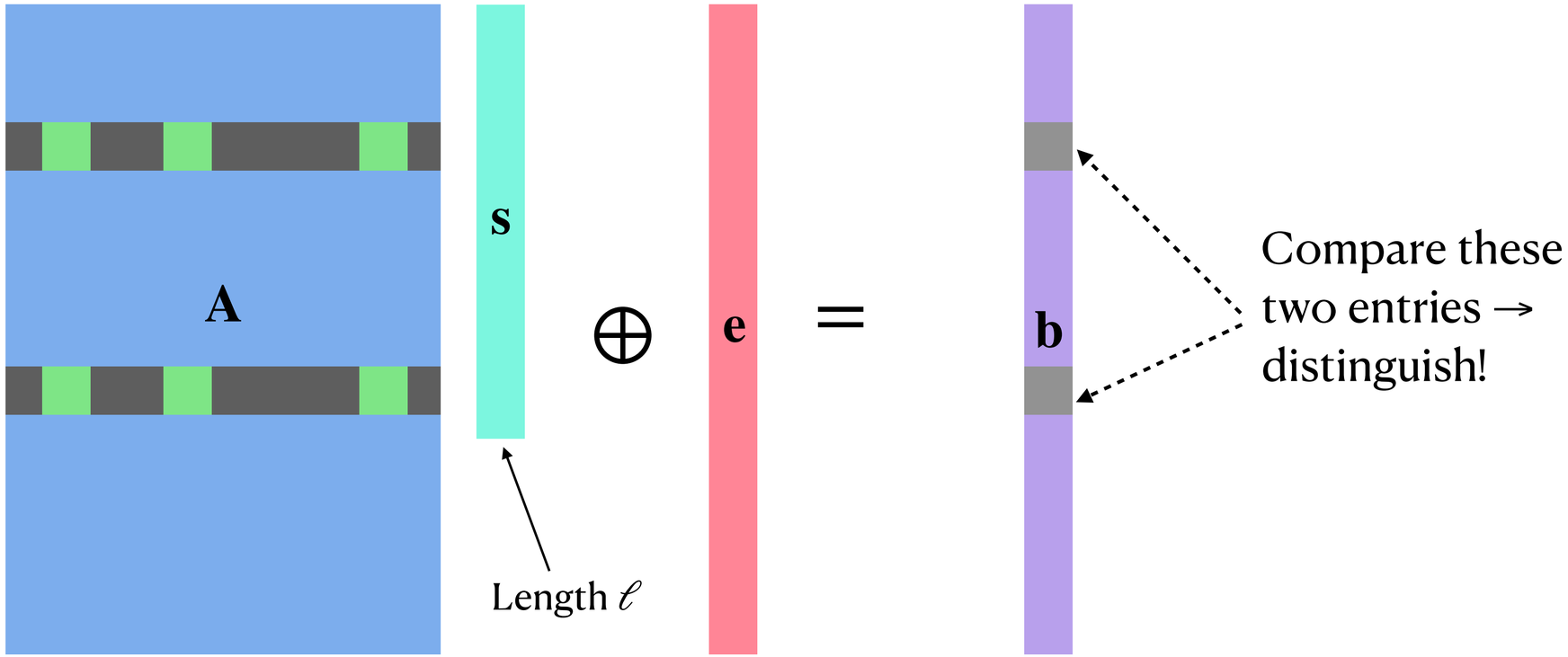
Sublinear Efficiency (Fast-XIO):

- We don't even have enough time to sample $\mathbf{e}$!
- Instead: *implicitly* sample $\mathbf{e}$ as a list of nonzero positions
- Can construct $\{\mathbf{U_i}, \mathbf{V_i}\}$ directly from this

But... Sparse LPN is *false.*

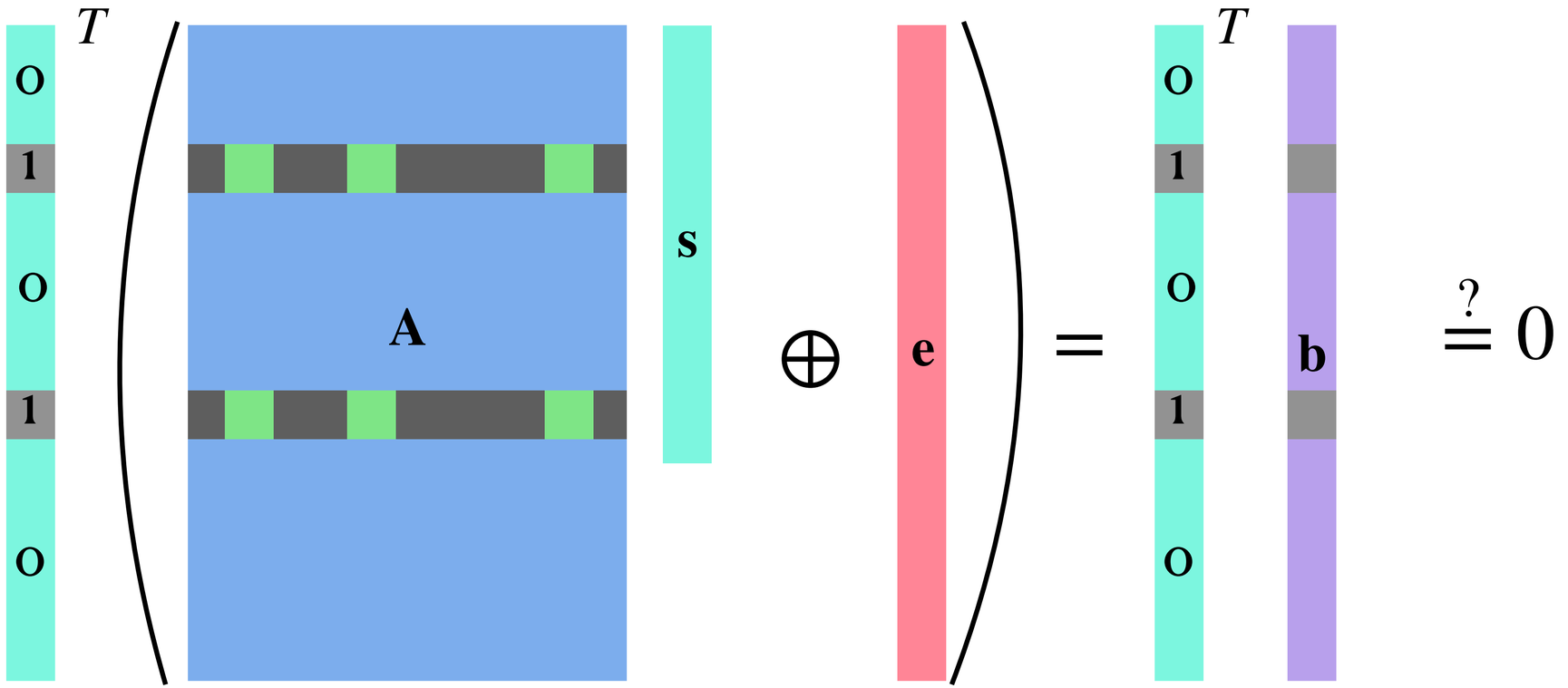
A Simple Attack

- With $1/\ell^{O(t)} = 1/\mathsf{poly}(\ell)$ probability, some two rows of $\mathbf{A}$ are the same
- Distinguisher succeeds w.p. $1/\mathsf{poly}(\ell)$



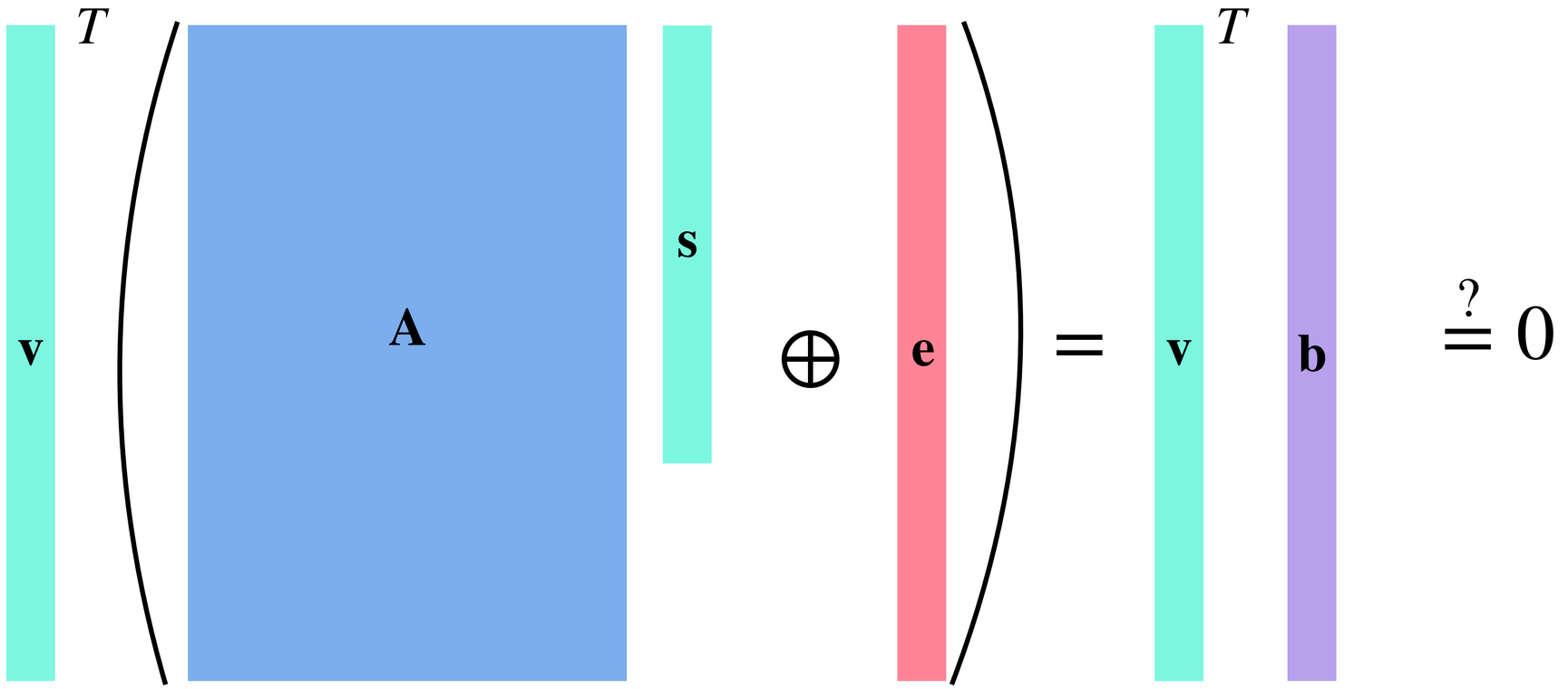
A Simple Attack

- With $1/\ell^{O(t)} = 1/\mathrm{poly}(\ell)$ probability, some two rows of $\mathbf{A}$ are the same
- Can also view this as a "linear test"



Linear Test Framework

- Dual distance $\mathsf{dd}(\mathbf{A})$: minimum Hamming weight of $\left\{\mathbf{v} \neq \mathbf{0} : \mathbf{v}^{\mathbf{T}}\mathbf{A} \equiv 0 \pmod 2\right\}$
- If $\mathsf{dd}(\mathbf{A}) \geq \ell^{\Omega(1)}$, then linear tests only achieve sub-exponential advantage
- This is true w.p. $1 - 1/\mathsf{poly}(\ell)$ over $\mathbf{A}$, but this is insufficient



What Now?

- If we only want *negligibly* secure XIO, then there exists a sampler for $\mathbf{A}$ by AK23 that is negligibly secure against linear tests
  - Insufficient to get to IO
- We don't know of such an efficient sampler for sub-exponential security
  - Option 1: put $\mathbf{A}$ in a crs → get an IO construction that is (plausibly) sub-exponentially secure with $1 - 1/\mathsf{poly}$ probability over the crs
  - Option 2: sample many $\mathbf{A}$ matrices, and somehow stitch them together so that only one needs to be sub-exponentially secure

Endgame: Functional Encryption Combiners

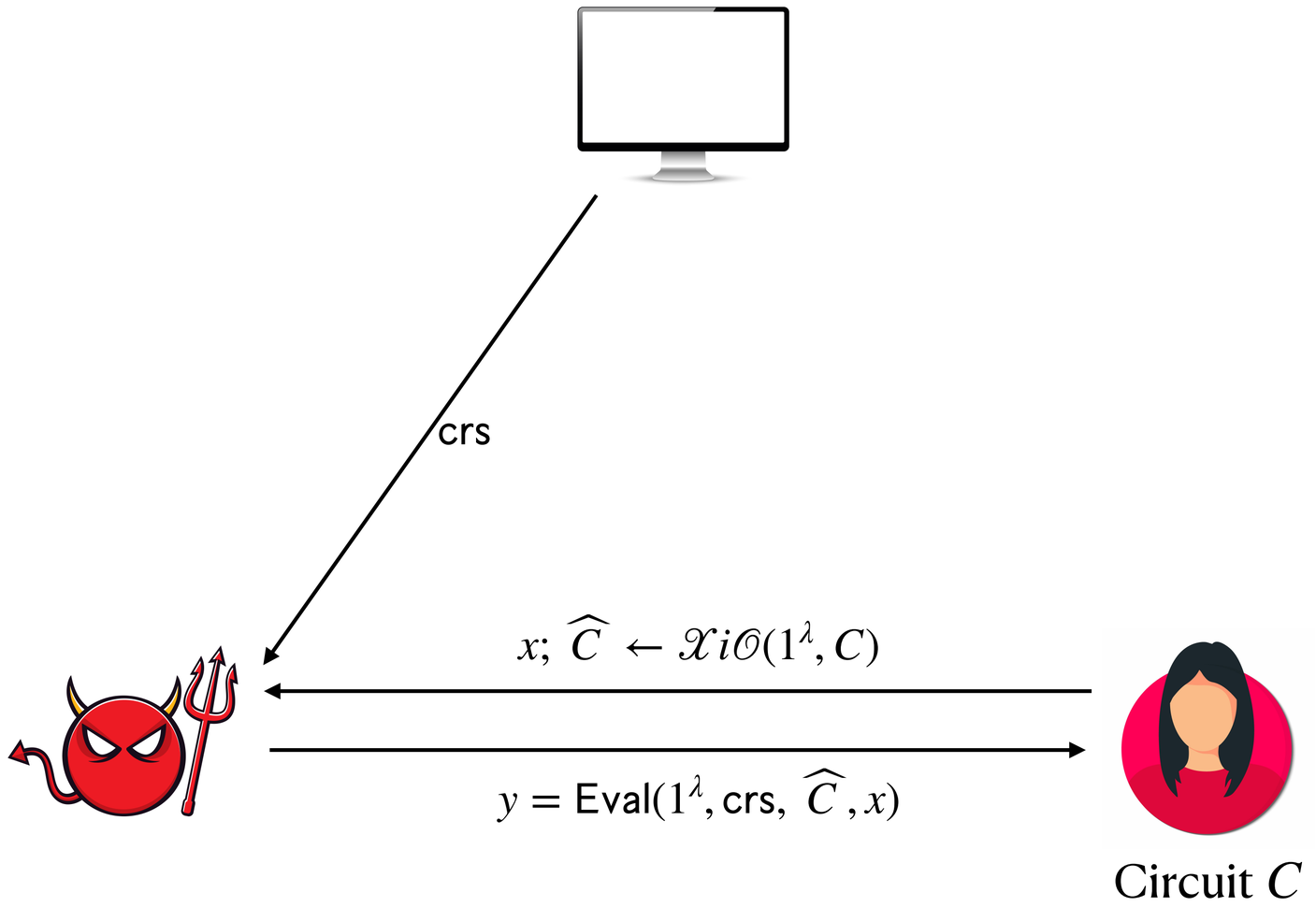
(Or: "Building Crypto from Slightly False Assumptions")

Wait, what?

Can't we combine $\mathbf{A}$ matrices directly?

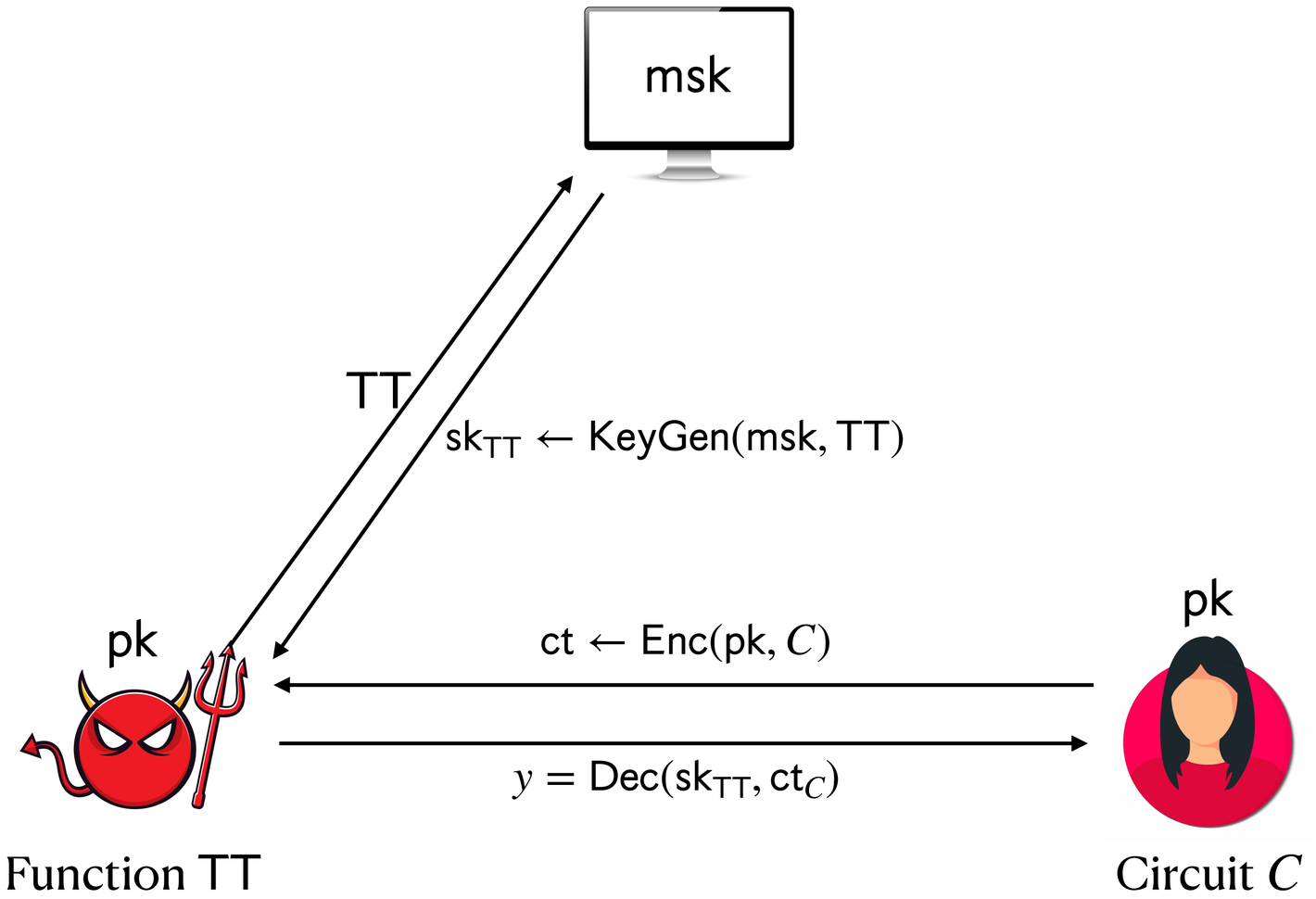
- We are using $\mathbf{A}$ to generate pseudorandom bits
- Why not sample $\mathbf{A_1}, \ldots, \mathbf{A_k}$ and output $\bigoplus_{i=1}^{k} \mathsf{Eval}(\mathbf{A_i}, \mathsf{seed}_i)$?
- The catch: XOR is not low-degree over $\mathbb{Z}$
  - This needs degree $k \to$ must have $k = O(1)$ (for compatibility with Wee20 PHFE)
  - And now this doesn't really improve security
- *But hold this thought...*

Context Switch: XIO → FE



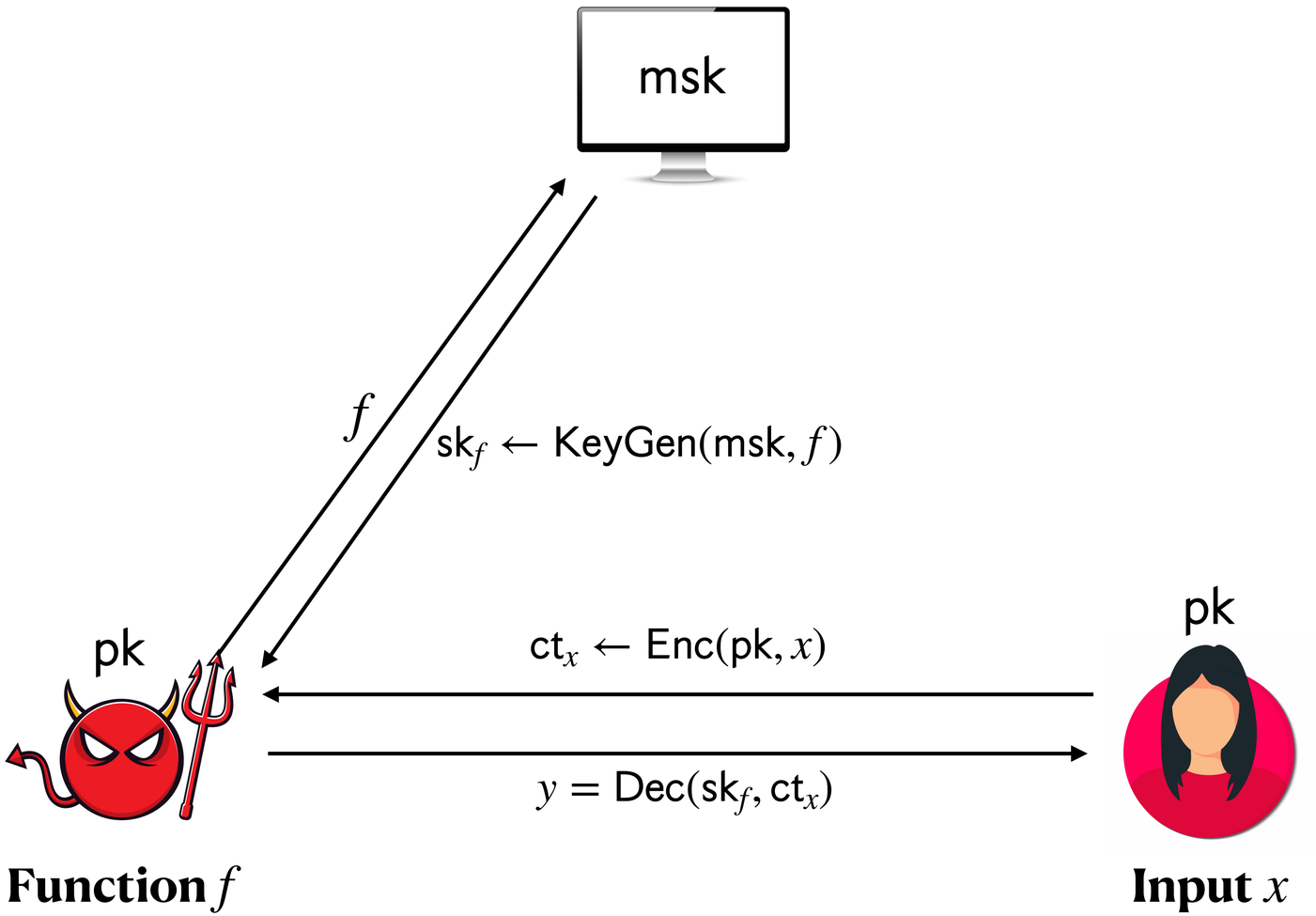
- We framed this construction as XIO...

Context Switch: XIO → FE



- We framed this construction as XIO...
- ... but we really did something that looks like FE for the TT function

Context Switch: XIO → FE

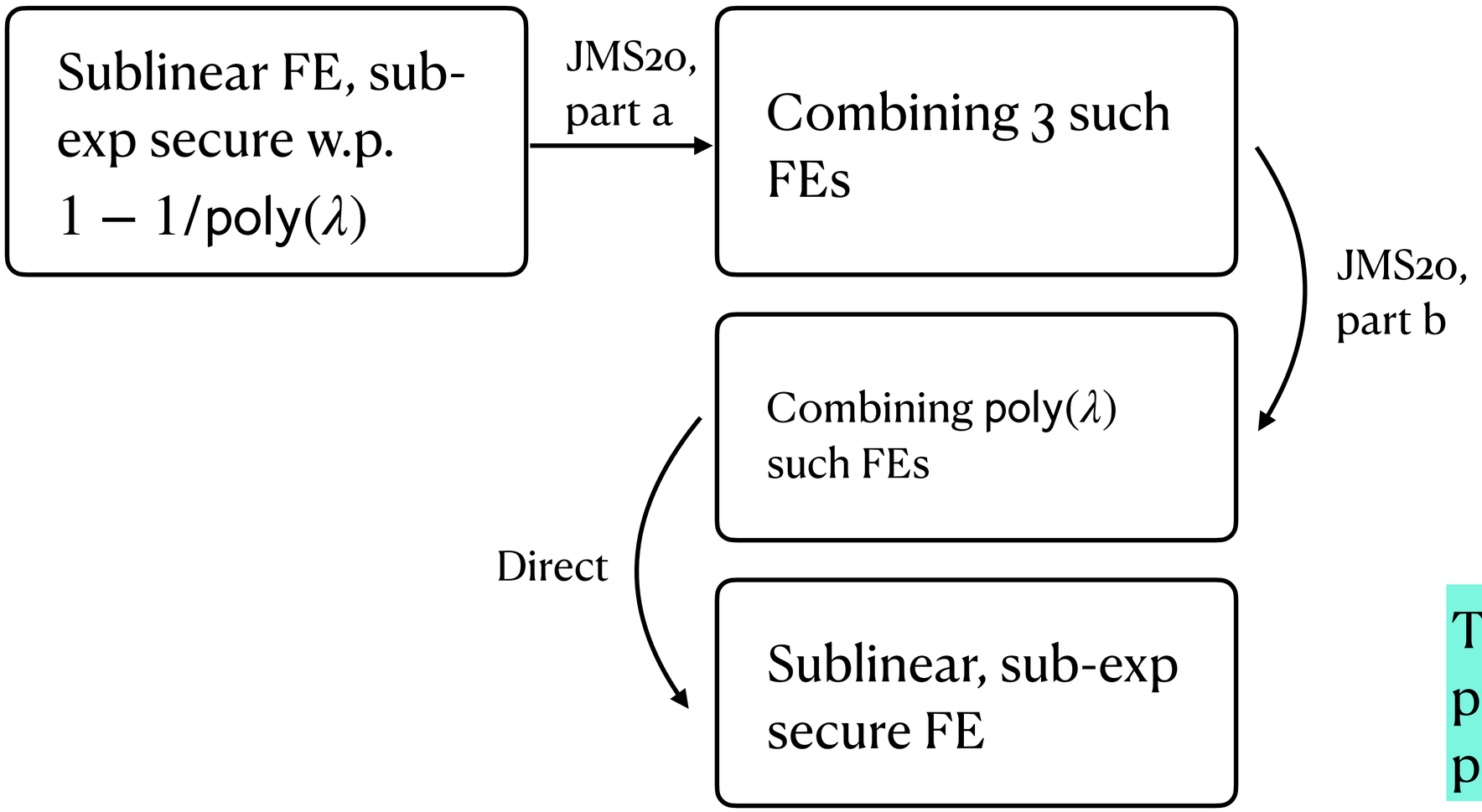


- We framed this construction as XIO...
- ... but we really did something that looks like FE for the TT function
- **This same construction gets FE for any circuit!**
- Compression requirement: $|\mathsf{ct}_x| \leq |f|^{1-\epsilon} \cdot \mathsf{poly}(\lambda)$

JMS20 FE Combiners

- Suppose we have several FE candidates $\mathsf{FE}_1, \mathsf{FE}_2, \ldots, \mathsf{FE}_k$, but only one is secure and we don't know which
- JMS20: these can be generically combined into a CombinedFE that is definitely secure!
- If we set $k = \mathsf{poly}(\lambda)$, we would get sub-exponential security!
- Q: But does JMS20 preserve sublinear efficiency?
- A: No, but there is a simple white-box workaround :)

JMS20 FE Combiner



TLDR: part b preserves sublinearity, part a does not

Fix for JMS20, Part a

Back to XORing PRGs

- The original naive XOR idea was actually fine for combining $O(1)$ many $\mathbf{A}$ matrices!
- Final construction:
  - Sample independent Sparse LPN matrices $\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}$, put them in the sky
  - In CombFE: use

$$(\mathsf{seed}_1 \,||\, \mathsf{seed}_2 \,||\, \mathsf{seed}_3) \mapsto \mathsf{Eval}(\mathbf{A_1}, \mathsf{seed}_1) \oplus \mathsf{Eval}(\mathbf{A_2}, \mathsf{seed}_2) \oplus \mathsf{Eval}(\mathbf{A_3}, \mathsf{seed}_3)$$

  as our (S)PRG!
- If at least one of $\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}$ is secure, so is the whole thing

FE Bootstrapping: Summary

Sublinear FE, sub-exp secure w.p. $1 - 1/\text{poly}(\lambda)$

XORing PRGs

Combining 3 such FEs, preserving sublinearity

JMS20, part b

Combining $\text{poly}(\lambda)$ such FEs, preserving sublinearity

Direct

Sublinear, sub-exp secure FE

AJ15, BV15, KNT22

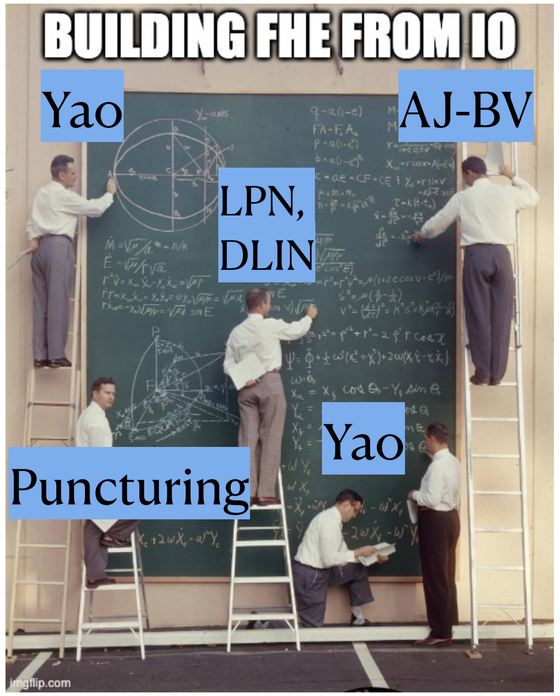
$i\mathcal{O}$!

Future Directions

- Minimal assumptions for IO?
- Post-quantum IO? (Bilinear maps completely broken)

Future Directions

- Minimal assumptions for IO?
- Post-quantum IO? (Bilinear maps completely broken)
- Can we get the applications (FHE, SNARGS, etc.) from these assumptions directly, without jumping through so many hoops?



Future Directions

- Minimal assumptions for IO?
- Post-quantum IO? (Bilinear maps completely broken)
- Can we get the applications (FHE, SNARGS, etc.) from these assumptions directly, without jumping through so many hoops?
- Cryptanalysis of “Goldreich + 1/poly noise”
- What other crypto can we build from LFN?