LogVOLE

Chosen-input VOLE with polylog communication for ZK and Beyond

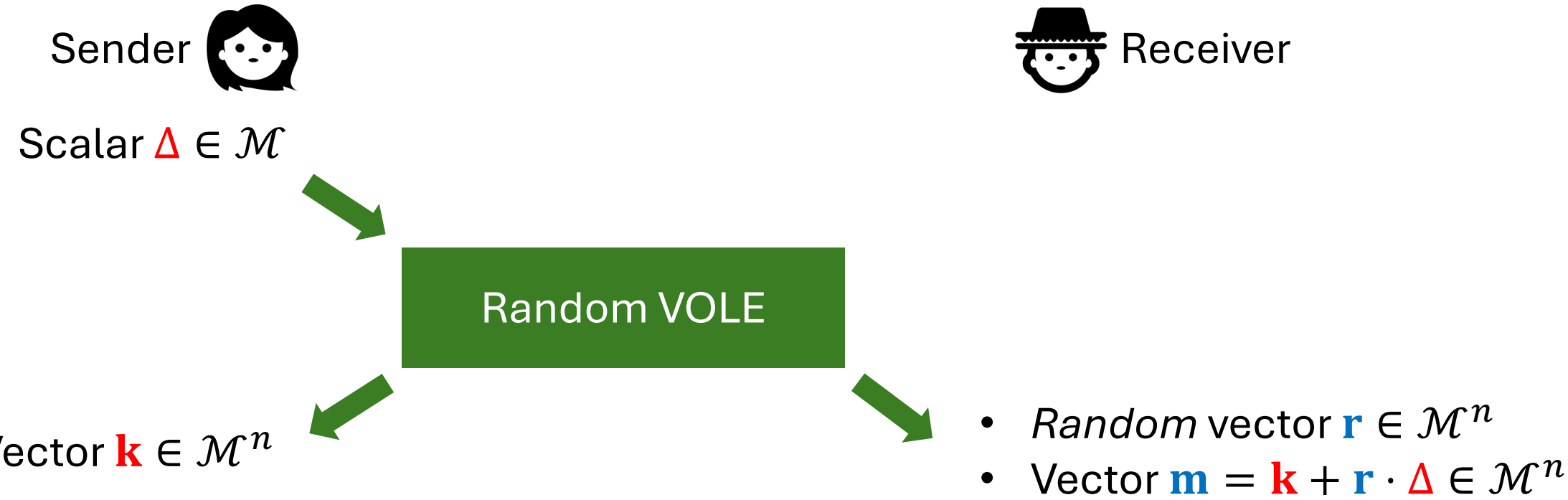
Lucien K. L. Ng (Georgia Tech)

Peter Rindal (Category Labs)

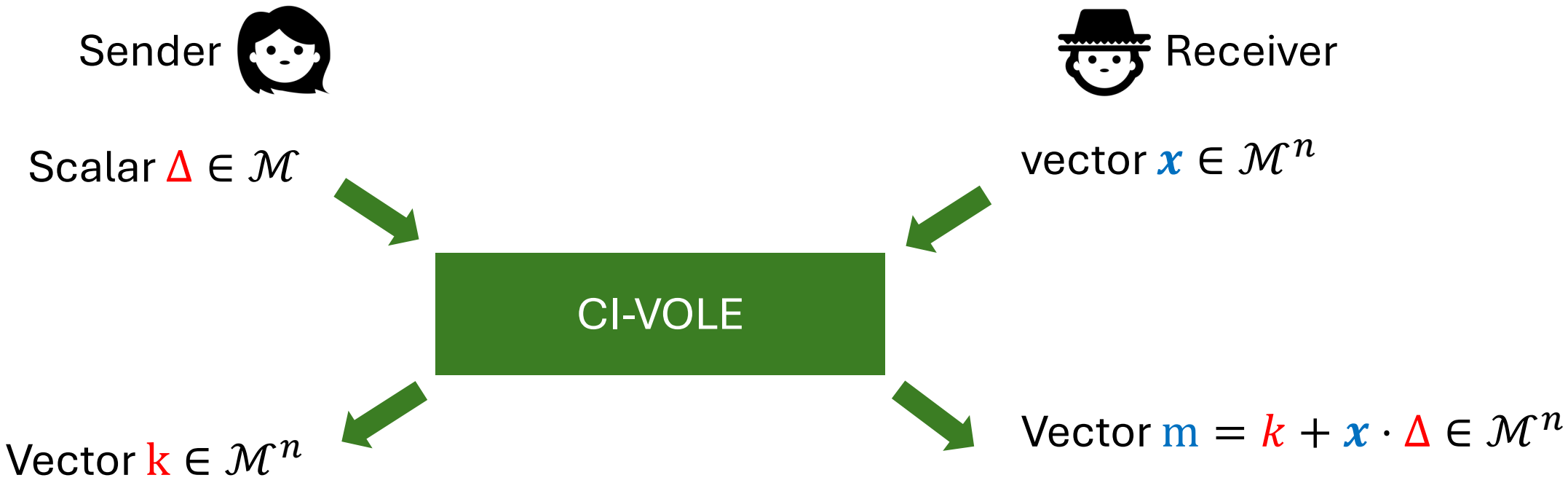
Akash Shah (UCLA ➔ J.P. Morgan)

(Random) VOLE

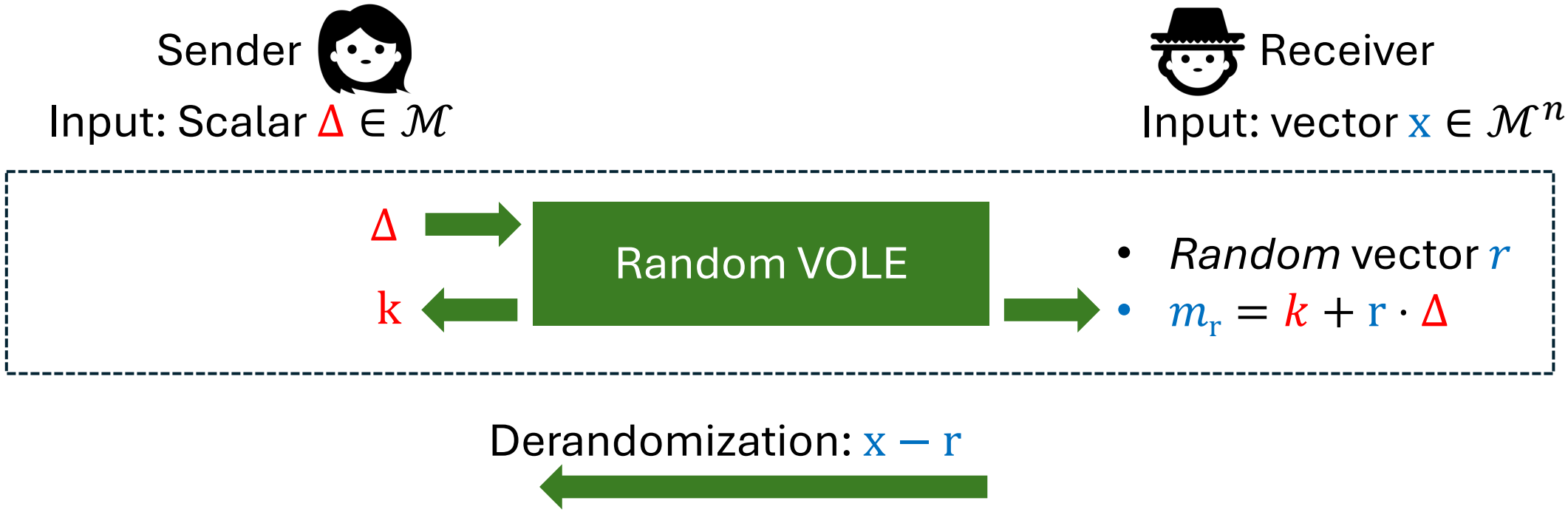
- VOLE: **V**ector **O**blivious **L**inear **E**valuation
  - [BCGI18, BCG+19b, SGRR19, BCG+19a, YWL+20, CRR21, RRT23]



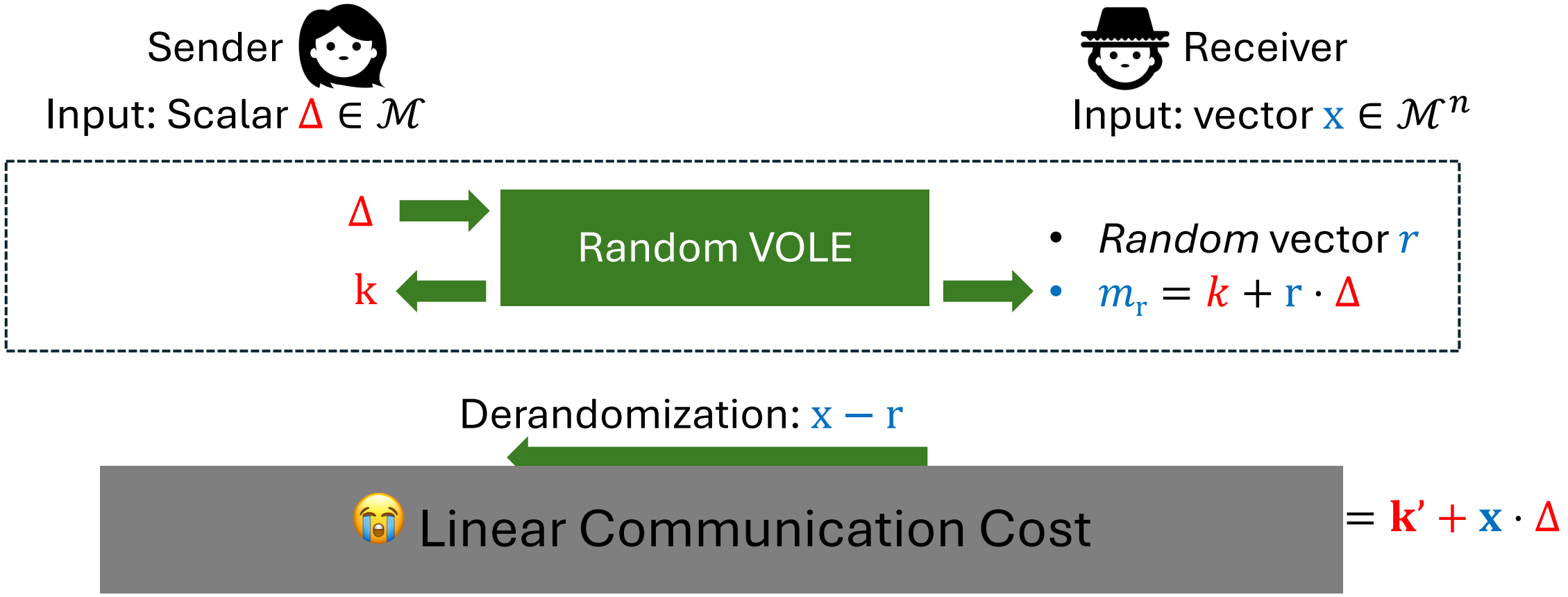
Ours: Chosen-Input (CI-)VOLE



Prior: Random VOLE ⇒ CI-VOLE



Prior: Random VOLE ⇒ CI-VOLE



Communication becoming a bottleneck

- Random VOLE can be used for zero-knowledge proofs
- Example: Zero-knowledge GPT Inference

Scheme	Prover	Verifier	Local Gen.	Size	Transfer
zk-SNARK-based	2.84h	24.43s	2.85h	15.6KB	0.10s
VOLE-based	136.1s	62.85s	198.95s	4.37GB	88.17s

Table 6: GPT-2 inference costs reported in Table 5 of Lu *et al.* [LWQ+24].

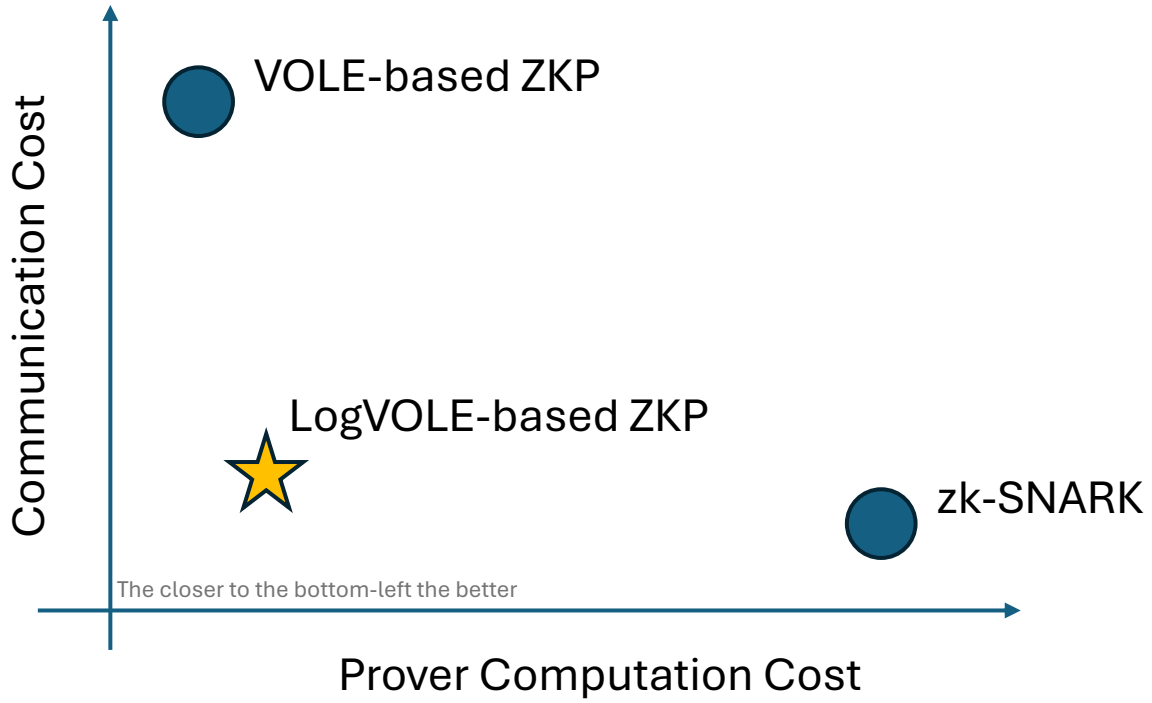
- Network Bandwidth: 500 Mbps

Can we reduce the communication cost of CI-VOLE while maintaining excellent concrete efficiency?

Yes. We propose LogVOLE

Applications: IT-MAC-based ZKP

- LogVOLE ⇒ ZKP with proof size = $O\left(\lambda \log^2(|C|)\right)$
  - vs. Random-VOLE-based ZKP's $O(\lambda\ |C|)$
  - |C| is the size of the verification circuit. λ = 128 is the security parameter
- LogVOLE has concrete computation efficiency



Concrete Proof size

- Example: Multiplying two 1024 x 1024 matrices

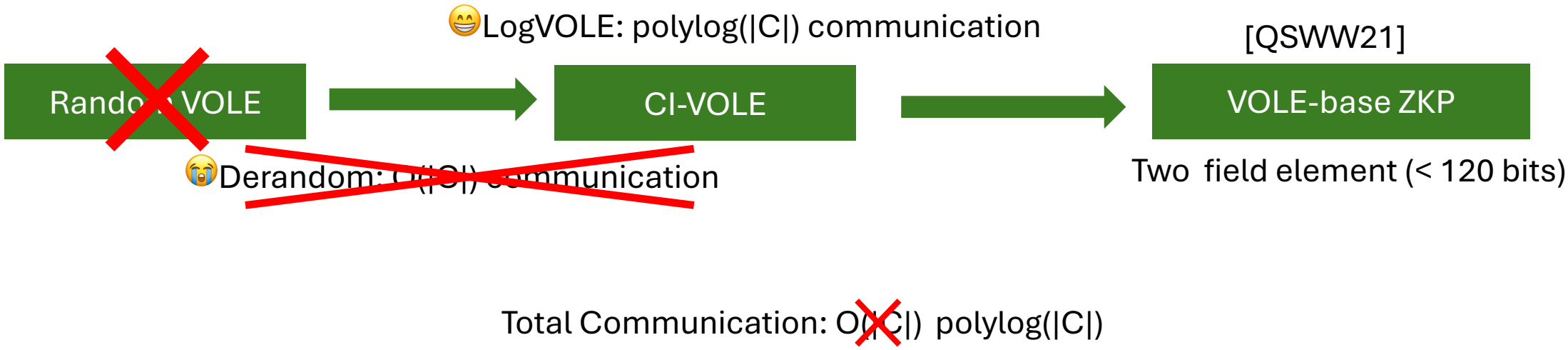
Scheme	Run Time	Comm.	ZK Genre
Spartan [48]	$\geq$ 5000 s	$\leq$ 100 KB	zk-SNARK
Virgo [55]	357 s	221 KB	zk-SNARK
Wolverine [50]	1627 s	30.7 GB*	IT-MAC
Mac'n'Cheese [6]	2684 s	23.3 GB*	IT-MAC
QuickSilver [52]	10 s	22.7 MB*	IT-MAC
JesseQ [41]	7 s	15.1 MB*	IT-MAC
Ours	4.2 s	226 KB	IT-MAC

How CI-VOLE save communication?

Total Communication: O(|C|)

[QSWW21]: Yang et al., Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. CCS 21

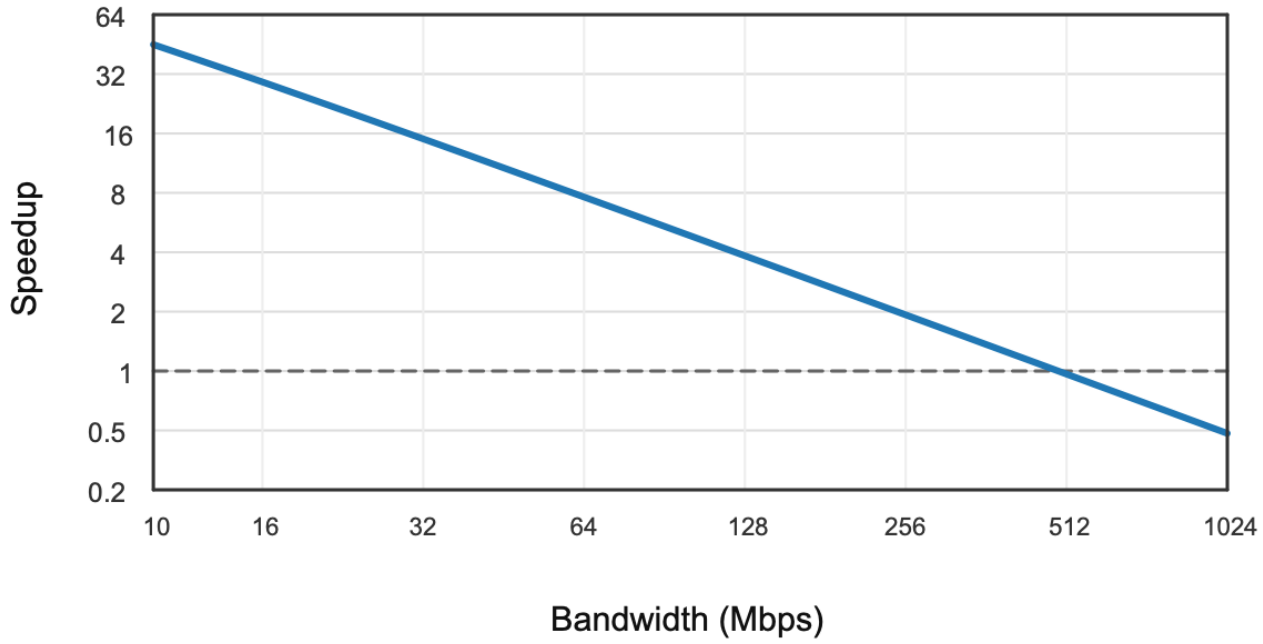
How CI-VOLE save communication?



[QSWW21]: Yang et al., Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. CCS 21

Our Runtime Speedup for ZKP

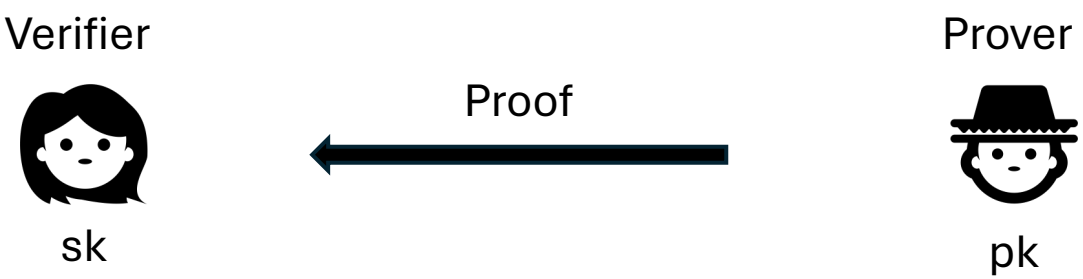
- Comparing against prior Random-VOLE-based ZKP
  - To their advantage, we only consider their de-randomization communication
  - We ignore their computation cost, and assume our throughput is 10 M/s



Upgrade: Non-interactive Public-Key ZKP

- Setup: Verifier computes *sk* and publishes *pk*

- Online:

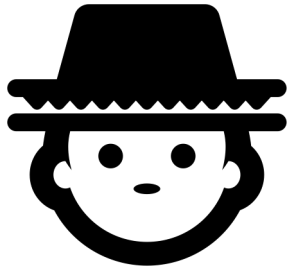


One-shot Communication + Good Prover Efficiency!

Another Application: Garbled Input Transfer

Garbler

Evaluator



Offline: Garbling Material M →

Online: Getting Input Labels ↔

Communication: $O(\lambda |x_G|) + O(\lambda |x_E|)$

😄 Constant Rounds

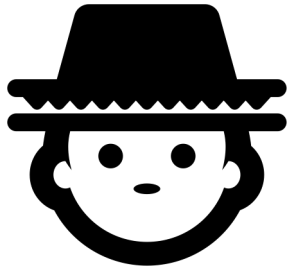
😄 Symmetric Key Primitives

😄 Authenticity

Another Application: Garbled Input Transfer

Garbler

Evaluator



Offline: Garbling Material M →

Online: Getting Input Labels ↔

Communication: ~~$O(\lambda|x_G|)$~~ + ~~$O(\lambda|x_E|)$~~

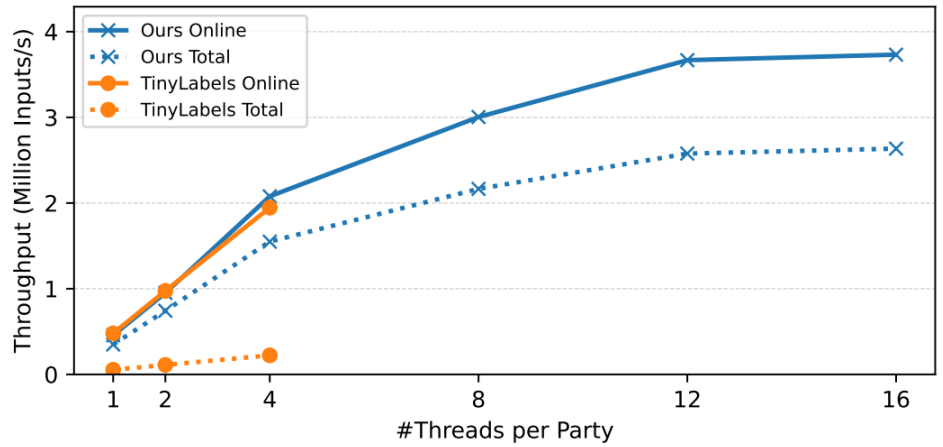
LogVOLE: $O(|x_G| + \lambda\ \mathrm{polylog}(|x|))$

😄 Constant Rounds

😄 Symmetric Key Primitives

😄 Authenticity

Concrete Performance



Garbled Input Compress Throughput vs. Thread

Scheme	Setup	Per-Query $N = 699{,}050$	Per-Query $N = 11{,}184{,}810$
Ours	38 MB (58×)	226 KB (2.3×)	226 KB (37.1×)
TinyLabel [24]	2.2 GB*	524 KB	8.39 MB

Communication Cost Reduction

How to apply LogVOLE for Garbled Inputs

- Free-XOR Garbled Circuit Label for an input wire with wire value x:

$$\text{Label}_x = K + x\,\Delta$$

Garbler (G)

Evaluator (E)

VOLE relation!

Comparing against Other Garbled Input Transfer

Scheme	Assumptions	One-Time Setup Communication	Per-Query Communication	Computation	Concrete Efficiency
Naive	–	–	$\lvert\mathbf{x}\rvert \cdot \lambda$	$O(\lvert\mathbf{x}\rvert)$ Hashing	✓
[5]	DDH	$O_\lambda(\lvert\mathbf{x}\rvert)$	$\lvert\mathbf{x}\rvert + \frac{\lvert\mathbf{x}\rvert}{\lambda}$	$O(\lvert\mathbf{x}\rvert \cdot \mathrm{poly}(\lambda))$ Mult_G	×
[5]	(R)LWE	$O_\lambda(\lvert\mathbf{x}\rvert)$	$\lvert\mathbf{x}\rvert + \frac{\lvert\mathbf{x}\rvert}{\lambda}$	$O(\lvert\mathbf{x}\rvert \cdot \mathrm{poly}(\lambda))$ Add_q	×
[4]	Bilinear DDH	$O_\lambda(\lvert\mathbf{x}\rvert^2)$	$\lvert\mathbf{x}\rvert + \frac{\lvert\mathbf{x}\rvert}{\lambda}$	$O(\lvert\mathbf{x}\rvert\lambda)$ Mult_G + $O(\lvert\mathbf{x}\rvert)$ Bilinear ops	×
[4]	iO + SSBH	–	$\lvert\mathbf{x}\rvert + \mathrm{poly}(\log\lvert\mathbf{x}\rvert, \lambda)$	$O(\lvert\mathbf{x}\rvert \cdot \mathrm{poly}(\lambda))$ iO-eval	×
[4]	RSA	–	$\lvert\mathbf{x}\rvert + \mathrm{poly}(\lambda)$	$O(\lvert\mathbf{x}\rvert \log\lvert\mathbf{x}\rvert)$ $\mathrm{Mult}_{\mathrm{RSA}}$	×
[1]	LWE	$\tilde{O}(\lvert\mathbf{x}\rvert \cdot \lambda^5)$	$\lvert\mathbf{x}\rvert + \tilde{O}(\lambda^2(\log\lvert\mathbf{x}\rvert + \lambda))$	$\tilde{O}(\lvert\mathbf{x}\rvert \cdot \lambda^4)$ $\mathrm{Mult}_{\mathbb{Z}}$	×
TinyLabel [24]	RLWE	$O_\lambda(\lvert\mathbf{x}\rvert)$	Theory: $\lvert\mathbf{x}\rvert + \mathrm{poly}(\lambda)$ Concrete: $6\lvert\mathbf{x}\rvert + \mathrm{poly}(\lambda)$	$O(\lvert\mathbf{x}\rvert \log\lvert\mathbf{x}\rvert/n)$ $\mathrm{Mult}_{\mathcal{R}}$	✓
Ours	RLWE	$O(\lambda \log^2\lvert\mathbf{x}\rvert)$	$O(\lambda \log^2\lvert\mathbf{x}\rvert)$	$O(\lvert\mathbf{x}\rvert/n)$ $\mathrm{Mult}_{\mathcal{R}}$	✓

Table 2: Comparison of garbled-input compression schemes.

Comparing against Other VOLE Schemes

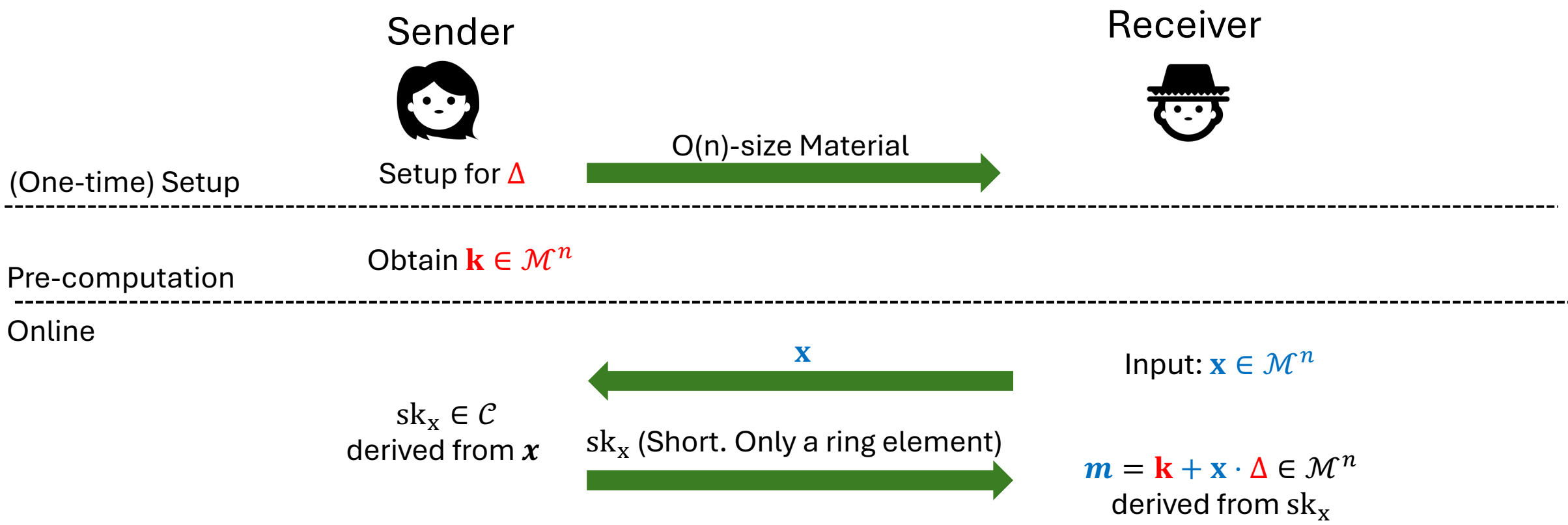
Scheme	Functionality	Assumptions	Total Communication	Computation	Support Non-Inter.	Concrete Efficiency
[12, 14, 15, 46, 47]	Random-VOLE	LPN / code-based	Random: $o(\|\mathbf{x}\|) + \text{poly}(\lambda)$ Chosen $\mathbf{x}$: $\Theta(\|\mathbf{x}\|) + \text{poly}(\lambda)$	$O(\|\mathbf{x}\|)$ Efficient Comp.	×	✓
[2]	CI-VOLE	DCR, QR, Power-DDH, LWE	$O_\lambda(\|\mathbf{x}\|^{2/3})$	$O_\lambda(\|\mathbf{x}\|^{4/3})$ $\text{Mult}_G/\text{Mult}_\mathbb{Z}$	✓	×
[2]	CI-VOLE	Power-RLWE	$O_\lambda(\|\mathbf{x}\|^{1/2})$	$O(\|\mathbf{x}\|^{3/2})$ $\text{Mult}_\mathcal{R}$	✓	×
[1]	CI-VOLE	LWE	$\tilde{O}(\lambda^5 \log \|\mathbf{x}\|)$	$\tilde{O}(\lambda^4 \|\mathbf{x}\|)$ $\text{Mult}_\mathbb{Z}$	✓	×
TinyLabel [24]	Staged CI-VOLE	RLWE	$O_\lambda(\|\mathbf{x}\|)$	$O(\|\mathbf{x}\| \log \|\mathbf{x}\|/n)$ $\text{Mult}_\mathcal{R}$	×	✓
Ours	Staged CI-VOLE	RLWE	$O(\lambda \log^2 \|\mathbf{x}\|)$	$O(\|\mathbf{x}\|/n)$ $\text{Mult}_\mathcal{R}$	×	✓
Ours	CI-VOLE	RLWE	$O(\lambda^3)$	$O(\|\mathbf{x}\|/n)$ $\text{Mult}_\mathcal{R}$	✓	✓

- $\tilde{O}(\cdot)$ denotes $O(\cdot)$ with $\text{poly}(\log \lambda)$ factors skipped. $O_\lambda(\cdot)$ denotes $O(\cdot)$ with $\text{poly}(\lambda)$ factors skipped. n is the degree of $\mathcal{R}$. m is the size of the message space $\mathcal{M}$.
- For the random-VOLE row, the first line counts only random-correlation generation. The second line is the end-to-end communication after adding the standard derandomization step for a chosen vector $\mathbf{x}$.
- "Supporting Non-Inter." means the protocol can be used in a non-interactive way, *e.g.*, by generating a long random correlation and then derandomizing it for multiple $\mathbf{x}$'s.
- For our protocol, the staged and non-interactive variants are listed separately because staging and non-interactive use are incompatible.

Table 1: Comparison of VOLE-related Protocols

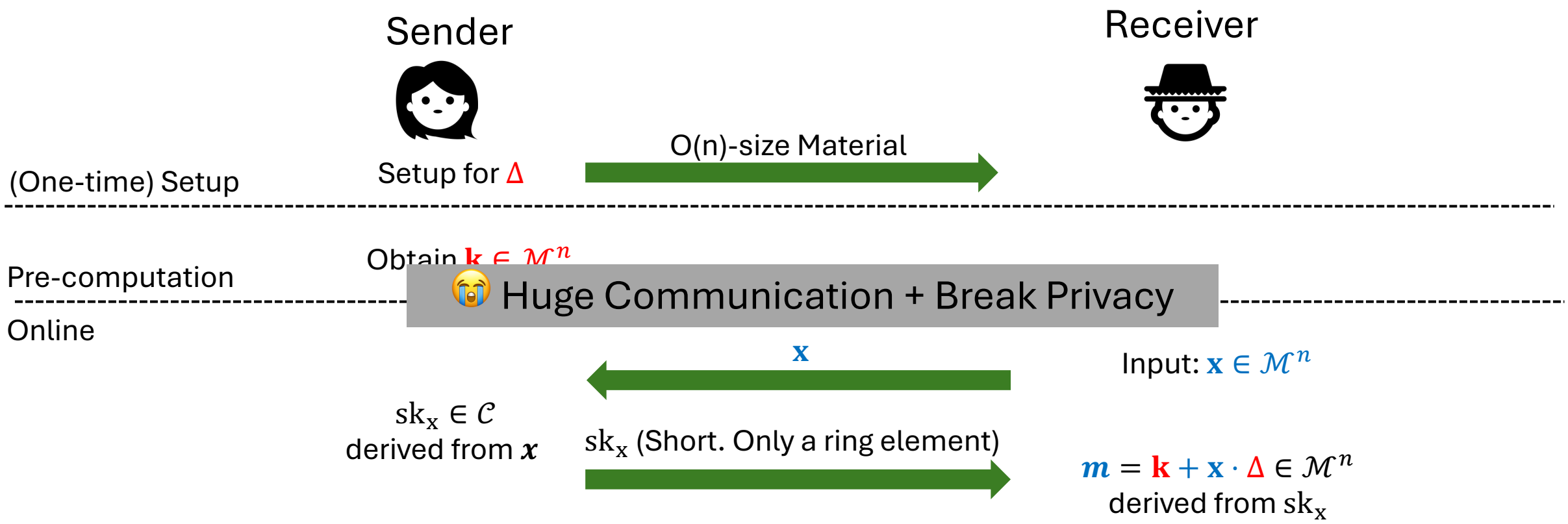
Preliminary

Revisiting TinyLabel (Eurocrypt'25)



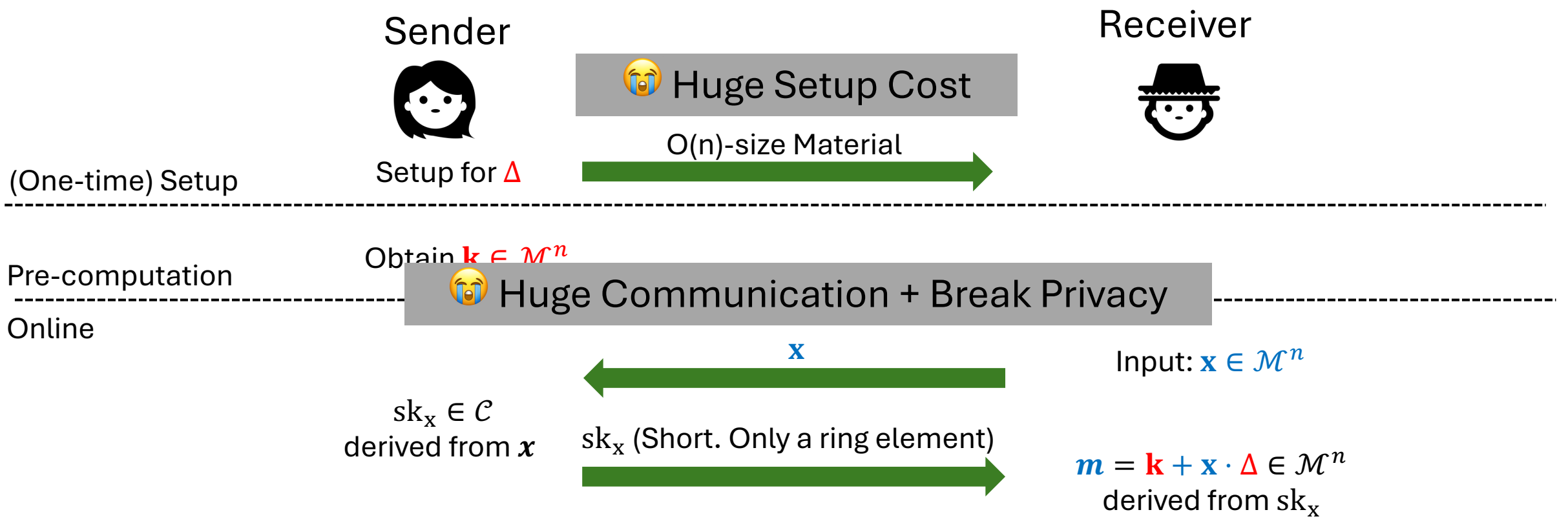
TinyLabel: a scheme for Garbled Input Transfer

Revisiting TinyLabel (Eurocrypt'25)



TinyLabel: a scheme for Garbled Input Transfer

Revisiting TinyLabel (Eurocrypt'25)



TinyLabel: a scheme for Garbled Input Transfer

Diving Into TinyLabel

Receiver

Input: $\mathbf{x} \in \mathcal{M}^n$

$$\text{LacE.Eval}(\text{ct}_{\text{lac}}, \mathbf{x}) \approx \mathbf{r} \cdot \text{d}_x - \Delta \cdot \mathbf{x}$$

- LacE: Laconic Encryption
- $\text{ct}_{\text{lac}} = \text{LacE.Enc}(\Delta; \mathbf{r})$ is a ciphertext prepared by Sender
- $\mathbf{r} \in \mathcal{M}^n$ is encryption randomness
- $\text{d}_x = \text{LacE.Digest}(x)$ is a short digest of x

Diving Into TinyLabel

Receiver

Input: $\mathbf{x} \in \mathcal{M}^n$

$$\text{LacE.Eval}(\text{ct}_{\text{lac}}, \mathbf{x}) \approx \mathbf{r} \cdot \text{d}_x - \Delta \cdot \mathbf{x}$$

🥲 $\mathbf{r} \cdot \text{d}_x$ is not pre-computable

- LacE: Laconic Encryption
- $\text{ct}_{\text{lac}} = \text{LacE.Enc}(\Delta; \mathbf{r})$ is a ciphertext prepared by Sender
- $\mathbf{r} \in \mathcal{M}^n$ is encryption randomness
- $\text{d}_x = \text{LacE.Digest}(x)$ is a short digest of x

TinyLabel’s Core Technical Contribution

$$\mathrm{LacE.Eval}(\mathrm{ct}_{\mathrm{lac}}, \mathbf{x}) \approx \mathbf{r} \cdot d_x - \Delta \cdot \mathbf{x}$$

*How can we replace the **x**-dependent mask **r** · d_x by a mask **k** that is fixed before **x** is chosen?*

TinyLabel's Solution

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← $\mathbf{x}$

$d_x = LacE.Digest(\mathbf{x})$ $d_x = LacE.Digest(\mathbf{x})$

$sk_k \leftarrow_\$ \mathcal{C}$
$\mathbf{k} \leftarrow_\$ \mathcal{M}^n$
$ct_k = LHE.Enc(sk_k, \mathbf{k})$
$sk_x = sk_r \cdot d_x + st_k$

ct_k, sk_x →

$\Delta \cdot x + k = \underbrace{LHE.Dec(sk_x, ct_r \cdot d_x + ct_k)}_{r \cdot d_x + k} - \underbrace{LacE.Eval(ct_{lac}, \mathbf{x})}_{\mathbf{r} \cdot d_x - \Delta \cdot \mathbf{x}}$

Note: LHE is Linear Homomorphic Encryption

TinyLabel's Solution

Sender

Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$

$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r

(One-time) Setup

Online

$\mathbf{x}$

$d_x = LacE.Digest(\mathbf{x})$

$d_x = LacE.Digest(\mathbf{x})$

$sk_k \leftarrow_\$ \mathcal{C}$

~~$\mathbf{k} \leftarrow_\$ \mathcal{M}^n$~~

~~$ct_k = LHE.Enc(sk_k, \mathbf{k})$~~

$sk_x = sk_r \cdot d_x + st_k$

$ct_k = H(seed)$

$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

~~ct_k~~, sk_x

$ct_k = H(seed)$

$\Delta \cdot x + k = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

$r \cdot d_x + k$ $\quad$ $\mathbf{r} \cdot d_x - \Delta \cdot \mathbf{x}$

Note: LHE is Linear Homomorphic Encryption

TinyLabel's Solution

Sender

Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$

$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r

(One-time) Setup

Online

$\mathbf{x}$

$d_x = LacE.Digest(\mathbf{x})$ $\quad$ $d_x = LacE.Digest(\mathbf{x})$

😄Short key: only a ring element (polynomial)

$sk_k \leftarrow_\$ \mathcal{C}$

~~$\mathbf{k} \leftarrow_\$ \mathcal{M}^n$~~

~~$ct_k = LHE.Enc(sk_k, \mathbf{k})$~~

$sk_x = sk_r \cdot d_x + st_k$

~~ct_k~~, sk_x

$ct_k = H(seed)$

$ct_k = H(seed)$

$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

$\Delta \cdot x + k = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

$r \cdot d_x + k$ $\qquad$ $\mathbf{r} \cdot d_x - \Delta \cdot \mathbf{x}$

Note: LHE is Linear Homomorphic Encryption

TinyLabel's Issues

Sender

Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

😭 O(|x|) linear-size Communication

ct_{lac}, ct_r

(One-time) Setup

Online

$\mathbf{x}$

$d_x = LacE.Digest(\mathbf{x})$ $\quad$ $d_x = LacE.Digest(\mathbf{x})$

😭 Linear Communication + Leaking **x**

$sk_k \leftarrow_\$ \mathcal{C}$
$ct_k = H(seed)$
$\mathbf{k} = LHE.Dec(ct_k, sk_k)$
$sk_x = sk_r \cdot d_x + st_k$

sk_x

$ct_k = H(seed)$

$\Delta \cdot x + k = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

Our LogVOLE Construction

Our 1st Idea: Removing Transmission of **x**

Sender 

 Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

ct_{lac}, ct_r

(One-time) Setup

Online

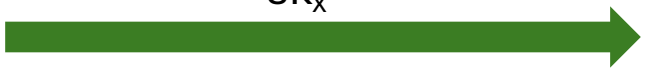
x

d_x = LacE.Digest(**x**)

d_x = LacE.Digest(**x**)

$sk_k \leftarrow_\$ \mathcal{C}$
ct_k= H(seed)
k = LHE.Dec(ct_k,sk_k)
$sk_x = sk_r \cdot d_x + sk_k$

sk_x



ct_k= H(seed)

Δ · x + **k** = LHE.Dec(sk_x, $ct_r \cdot d_x + ct_k$)
- LacE.Eval(ct_{lac}, **x**)

Our 1st Idea: Removing Transmission of **x**

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← $\mathbf{x}$

$d_x = LacE.Digest(\mathbf{x})$ $\quad$ $d_x = LacE.Digest(\mathbf{x})$

Observe:
x is not needed

$sk_k \leftarrow_\$ \mathcal{C}$
$ct_k = H(seed)$
$\mathbf{k} = LHE.Dec(ct_k, sk_k)$
$sk_x = sk_r \cdot d_x + sk_k$

sk_x →

$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k)$
$- LacE.Eval(ct_{lac}, \mathbf{x})$

Our 1st Idea: Removing Transmission of **x**

Sender 

 Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

ct_{lac}, ct_r

(One-time) Setup

Online

d’

d_x = LacE.Digest(**x**)
d’= Randomize(d_x)

Observe:
x is not needed

$sk_k \leftarrow_\$ \mathcal{C}$
ct_k= H(seed)
k = LHE.Dec(ct_k,sk_k)
sk’$_x$ = sk_r · d’ + sk_k

sk’$_x$

sk_x = Derandom(sk’$_x$)
ct_k= H(seed)

Δ · x + **k** = LHE.Dec(sk_x, ct_r · d_x + ct_k)
- LacE.Eval(ct_{lac}, **x**)

Our 1st Idea: Removing Transmission of **x**

Sender

Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r

(One-time) Setup

Online

d'

$d_x = LacE.Digest(\mathbf{x})$

$\ldots ize(d_x)$

Short Digest + Hide **x**

Observe:
x is not needed

$sk_k \leftarrow_\$ \mathcal{C}$
$ct_k = H(seed)$
$\mathbf{k} = LHE.Dec(ct_k, sk_k)$
$sk'_x = sk_r \cdot d' + sk_k$

sk'_x

$sk_x = Derandom(sk'_x)$
$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k)$
$- LacE.Eval(ct_{lac}, \mathbf{x})$

Our 1st Idea: Removing Transmission of **x**

Sender

Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

😭 Still linear-size communication

ct_{lac}, ct_r

(One-time) Setup

Online

d'

d_x = LacE.Digest(**x**)

ize(d_x)

😄 Short Digest + Hide **x**

$sk_k \leftarrow_\$ \mathcal{C}$
ct_k= H(seed)
k = LHE.Dec(ct_k,sk_k)
sk'$_x$ = sk_r · d' + sk_k

Observe:
x is not needed

sk'$_x$

sk_x = Derandom(sk'$_x$)
ct_k= H(seed)

Δ · x + **k** = LHE.Dec(sk_x, ct_r · d_x + ct_k)
- LacE.Eval(ct_{lac}, **x**)

Warmup: Balancing Trick

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$

$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

Input: $\mathbf{x = x_1 || x_2 || x_3 || \ldots || x_\mu}$

Repeat for i in {1, …, **μ**}

← d'

$d_{xi} = LacE.Digest(\mathbf{x_i})$

$d' = Randomize(d_{xi})$

$sk_{ki} \leftarrow_\$ \mathcal{C}$

$ct_k = H(seed)$

$\mathbf{k_i} = LHE.Dec(ct_k, sk_{ki})$

$sk'_{xi} = sk_r \cdot d' + sk_{ki}$

sk'_x →

$sk_x = Derandom(sk'_x)$

$ct_k = H(seed)$

$\Delta \cdot xi + ki = LHE.Dec(sk_x, ct_r \cdot d_{xi} + ct_k)$
$- LacE.Eval(ct_{lac}, \mathbf{xi})$

Warmup: Balancing Trick

Sender 

 Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

ct_{lac}, ct_r

(One-time) Setup

Online

Input: **x = x_1|| x_2 || x_3 || … || x_μ**

$|x_i| = \Theta\left(\sqrt{|x|}\right)$

Repeat for i in {1, …, **μ**}

d_{xi} = LacE.Digest(**x_i**)

d’= Randomize(d_{xi})

d’

$sk_{ki} \leftarrow_\$ \mathcal{C}$
ct_k= H(seed)
k_i = LHE.Dec(ct_k,sk_{ki})
sk’$_{xi}$ = sk_r · d’ + sk_{ki}

sk’$_x$

sk_x = Derandom(sk’$_x$)
ct_k= H(seed)

Δ · xi + ki = LHE.Dec(sk_x, ct_r · d_{xi} + ct_k)
- LacE.Eval(ct_{lac}, **xi**)

Warmup: Balancing Trick

Sender

Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

Setup Cost becomes $\Theta\left(\sqrt{|x|}\right)$

ct_{lac}, ct_r

(One-time) Setup

Online

Input: **x = x_1|| x_2 || x_3 || … || x_μ**

$|x_i| = \Theta\left(\sqrt{|x|}\right)$

Repeat for i in {1, …, **μ**}

d’

d_{xi} = LacE.Digest(**x_i**)

d’= Randomize(d_{xi})

$sk_{ki} \leftarrow_\$ \mathcal{C}$
ct_k= H(seed)
k_i = LHE.Dec(ct_k,sk_{ki})
sk’$_{xi}$ = sk_r · d’ + sk_{ki}

sk’$_x$

sk_x = Derandom(sk’$_x$)
ct_k= H(seed)

Δ · xi + ki = LHE.Dec(sk_x, ct_r · d_{xi} + ct_k)
- LacE.Eval(ct_{lac}, **xi**)

Similarity in Digests and Outputs

Sender

Receiver

Input: $x_1, x_2, x_3, \ldots, x_\mu$

$d_1, d_2, \ldots, d_\mu$

d_i = LacE.Digest(x_i) for i in {1, …, μ}

$$\begin{pmatrix} \mathrm{sk}_{\mathbf{x}_1} \\ \vdots \\ \mathrm{sk}_{\mathbf{x}_\mu} \end{pmatrix} = \mathrm{sk}_r \cdot \begin{pmatrix} d_{\mathbf{x}_1} \\ \vdots \\ d_{\mathbf{x}_\mu} \end{pmatrix} + \begin{pmatrix} \mathrm{sk}_{k,1} \\ \vdots \\ \mathrm{sk}_{k,\mu} \end{pmatrix} \in \mathcal{C}^\mu.$$

$sk_{x1}, \ldots, sk_{x\mu}$

$sk_{k,i} \Rightarrow k_i$ for all i

$(d_i, sk_{xi}) \Rightarrow m_i$ for i in {1, …, μ}

$$\begin{pmatrix} \mathbf{m}_{\mathbf{x}_1} \\ \vdots \\ \mathbf{m}_{\mathbf{x}_\mu} \end{pmatrix} = \Delta \cdot \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_\mu \end{pmatrix} + \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_\mu \end{pmatrix} \in \mathcal{M}^{w\mu}.$$

Similarity in Digests and Outputs

Sender

Receiver

Input: $x_1, x_2, x_3, \ldots, x_\mu$

$d_1, d_2, \ldots, d_\mu$

d_i = LacE.Digest(x_i) for i in {1, …, μ}

$$\begin{pmatrix} sk_{x_1} \\ \vdots \\ sk_{x_\mu} \end{pmatrix} = sk_r \cdot \begin{pmatrix} d_{x_1} \\ \vdots \\ d_{x_\mu} \end{pmatrix} + \begin{pmatrix} sk_{k,1} \\ \vdots \\ sk_{k,\mu} \end{pmatrix} \in \mathcal{C}^\mu.$$

$$\langle sk_r \cdot d_x \rangle$$

$sk_{x1}, \ldots, sk_{x\mu}$

$sk_{k,i} \Rightarrow k_i$ for all i

$(d_i, sk_{xi}) \Rightarrow m_i$ for i in {1, …, μ}

$$\begin{pmatrix} \mathbf{m}_{\mathbf{x}_1} \\ \vdots \\ \mathbf{m}_{\mathbf{x}_\mu} \end{pmatrix} = \Delta \cdot \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_\mu \end{pmatrix} + \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_\mu \end{pmatrix} \in \mathcal{M}^{w\mu}.$$

$$\langle \Delta \cdot \mathbf{x} \rangle$$

$\langle z \rangle$ is a subtractive secret share where Sender holds z^S and Receiver hold z^R such that $z^R - z^S = z$

Similarity in Digests and Outputs

Sender

Receiver

Input: $x_1, x_2, x_3, \ldots, x_\mu$

$$\begin{pmatrix} \mathrm{sk}_{\mathbf{x}_1} \\ \vdots \\ \mathrm{sk}_{\mathbf{x}_\mu} \end{pmatrix} = \mathrm{sk}_r \cdot \begin{pmatrix} d_{\mathbf{x}_1} \\ \vdots \\ d_{\mathbf{x}_\mu} \end{pmatrix} + \begin{pmatrix} \mathrm{sk}_{k,1} \\ \vdots \\ \mathrm{sk}_{k,\mu} \end{pmatrix} \in \mathcal{C}^\mu.$$

$\langle \mathrm{sk}_r \cdot \mathrm{d}_x \rangle$

$d_1, d_2, \ldots, d_\mu$

d_i = LacE.Digest(x_i) for i in {1, …, μ}

Shrink: w μ inputs ➔ μ digests

$sk_{x1}, \ldots, sk_{x\mu}$

$sk_{k,i} \Rightarrow k_i$ for all i

$(d_i, sk_{xi}) \Rightarrow m_i$ for i in {1, …, μ}

$$\begin{pmatrix} \mathbf{m}_{\mathbf{x}_1} \\ \vdots \\ \mathbf{m}_{\mathbf{x}_\mu} \end{pmatrix} = \Delta \cdot \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_\mu \end{pmatrix} + \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_\mu \end{pmatrix} \in \mathcal{M}^{w\mu}.$$

$\langle \Delta \cdot \mathbf{x} \rangle$

$\langle z \rangle$ is a subtractive secret share where Sender holds z^S and Receiver hold z^R such that $z^R - z^S = z$

Similarity in Digests and Outputs

Sender

Receiver

Input: $x_1, x_2, x_3, \ldots, x_\mu$

$$\begin{pmatrix} \mathrm{sk}_{\mathbf{x}_1} \\ \vdots \\ \mathrm{sk}_{\mathbf{x}_\mu} \end{pmatrix} = \mathrm{sk}_r \cdot \begin{pmatrix} d_{\mathbf{x}_1} \\ \vdots \\ d_{\mathbf{x}_\mu} \end{pmatrix} + \begin{pmatrix} \mathrm{sk}_{k,1} \\ \vdots \\ \mathrm{sk}_{k,\mu} \end{pmatrix} \in \mathcal{C}^\mu.$$

$\langle \mathrm{sk}_r \cdot \mathrm{d}_x \rangle$

$d_1, d_2, \ldots, d_\mu$

d_i = LacE.Digest(x_i) for i in {1, …, μ}

Shrink: w μ inputs ➔ μ digests

$sk_{x1}, \ldots, sk_{x\mu}$

$sk_{k,i} \Rightarrow k_i$ for all i

$(d_i, sk_{xi}) \Rightarrow m_i$ for i in {1, …, μ}

Expand: μ keys ➔ w μ outputs

$$\begin{pmatrix} \mathbf{m}_{\mathbf{x}_1} \\ \vdots \\ \mathbf{m}_{\mathbf{x}_\mu} \end{pmatrix} = \Delta \cdot \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_\mu \end{pmatrix} + \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_\mu \end{pmatrix} \in \mathcal{M}^{w\mu}.$$

$\langle \Delta \cdot \mathbf{x} \rangle$

$\langle z \rangle$ is a subtractive secret share where Sender holds z^S and Receiver hold z^R such that $z^R - z^S = z$

Similarity in Digests and Outputs

Sender

Receiver

Input: $x_1, x_2, x_3, \ldots, x_\mu$

$$\begin{pmatrix} \mathrm{sk}_{\mathbf{x}_1} \\ \vdots \\ \mathrm{sk}_{\mathbf{x}_\mu} \end{pmatrix} = \mathrm{sk}_r \cdot \begin{pmatrix} d_{\mathbf{x}_1} \\ \vdots \\ d_{\mathbf{x}_\mu} \end{pmatrix} + \begin{pmatrix} \mathrm{sk}_{k,1} \\ \vdots \\ \mathrm{sk}_{k,\mu} \end{pmatrix} \in \mathcal{C}^\mu.$$

$\langle \mathrm{sk}_r \cdot \mathrm{d}_x \rangle$

$d_1, d_2, \ldots, d_\mu$

d_i = LacE.Digest(x_i) for i in {1, …, μ}

Shrink: w μ inputs ➔ μ digests

Observation:
Shrink and Expand are local operations

$sk_{x1}, \ldots, sk_{x\mu}$

$sk_{k,i} \Rightarrow k_i$ for all i

$(d_i, sk_{xi}) \Rightarrow \mathbf{m_i}$ for i in {1, …, μ}

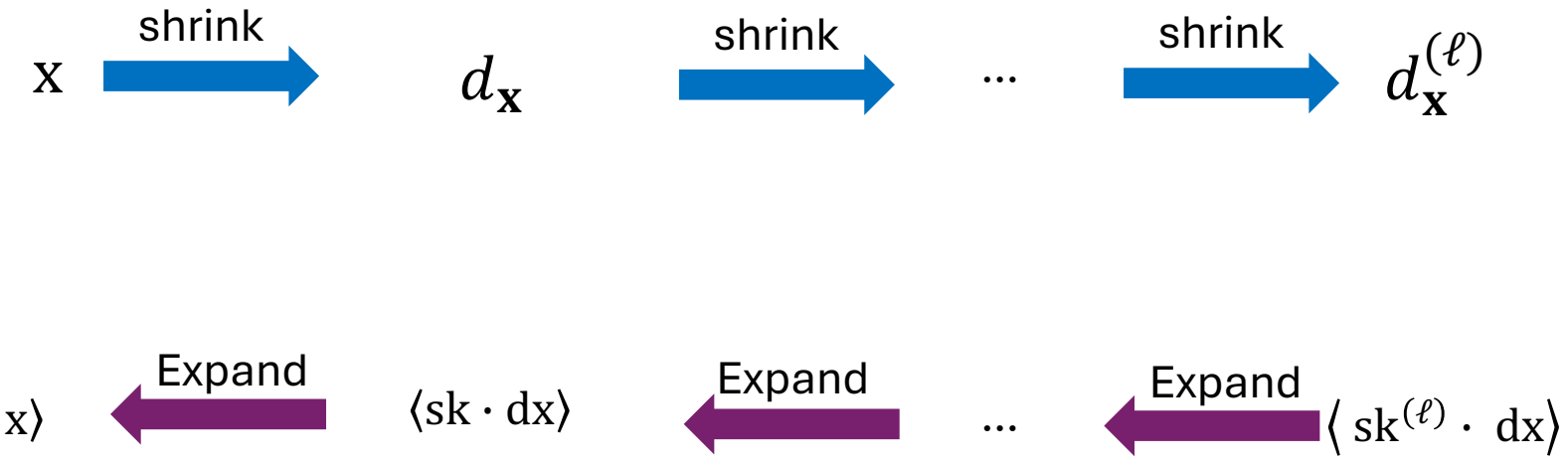
Expand: μ keys ➔ w μ outputs

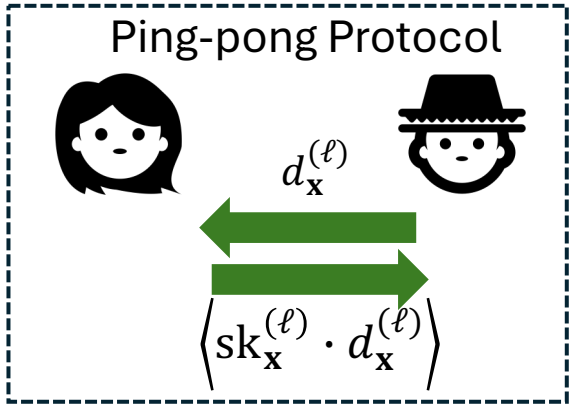
$$\begin{pmatrix} \mathbf{m}_{\mathbf{x}_1} \\ \vdots \\ \mathbf{m}_{\mathbf{x}_\mu} \end{pmatrix} = \Delta \cdot \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_\mu \end{pmatrix} + \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_\mu \end{pmatrix} \in \mathcal{M}^{w\mu}.$$

$\langle \Delta \cdot \mathbf{x} \rangle$

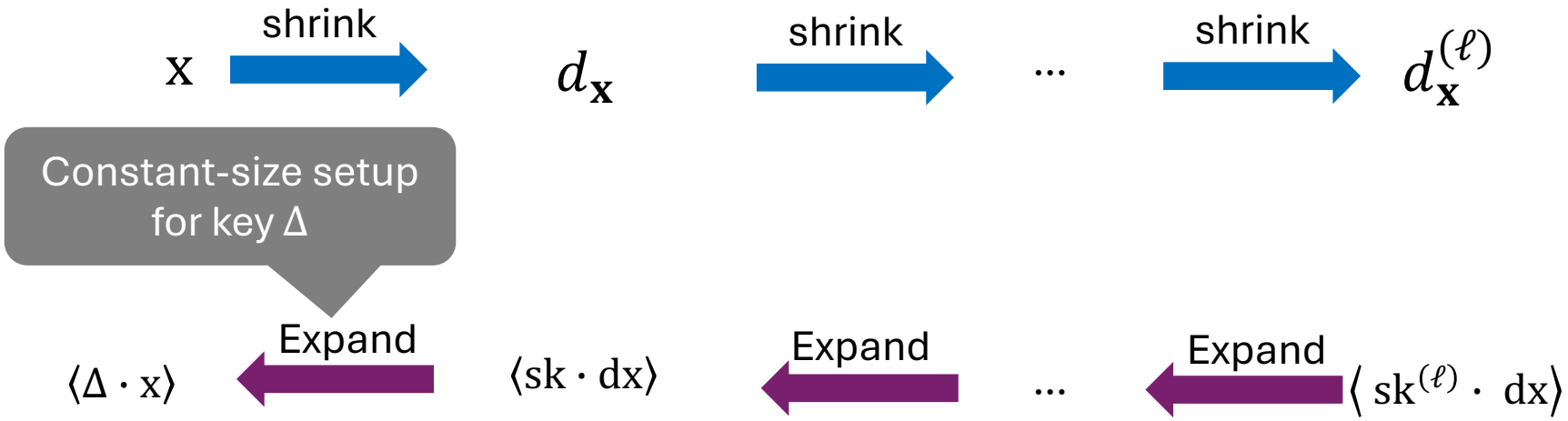
$\langle z \rangle$ is a subtractive secret share where Sender holds z^S and Receiver hold z^R such that $z^R - z^S = z$

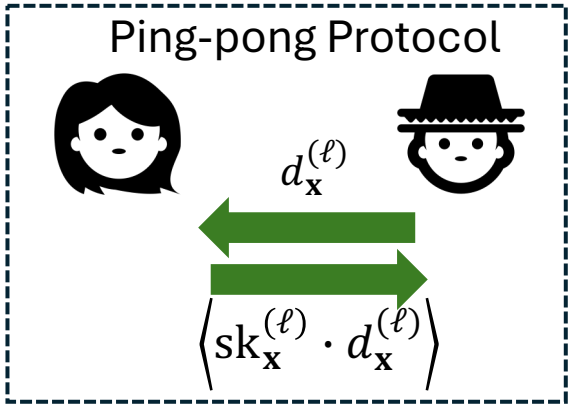
Recursive Construction



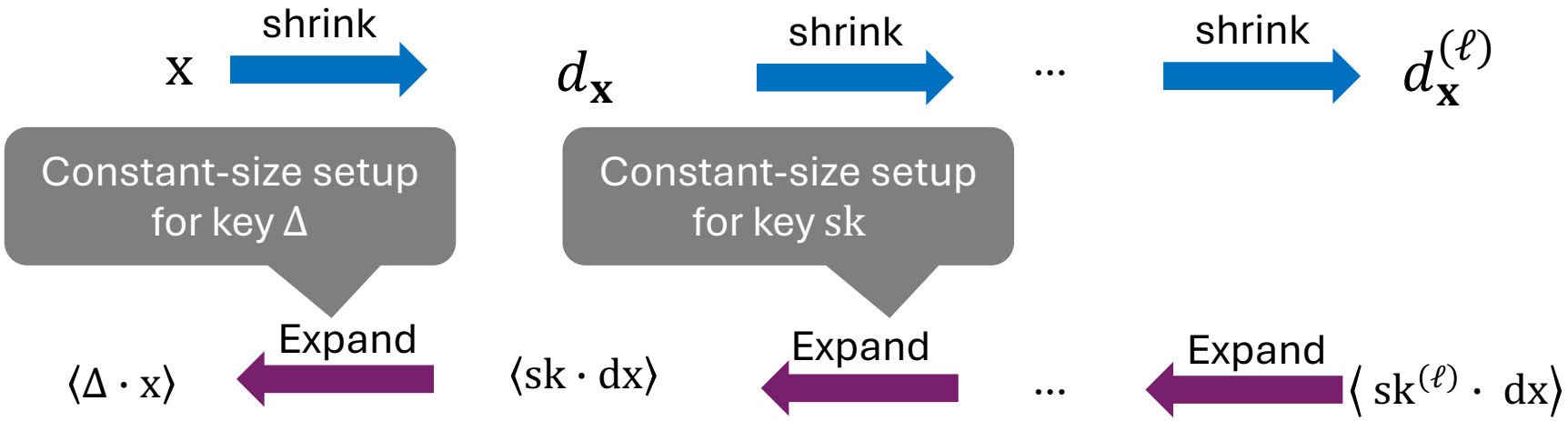


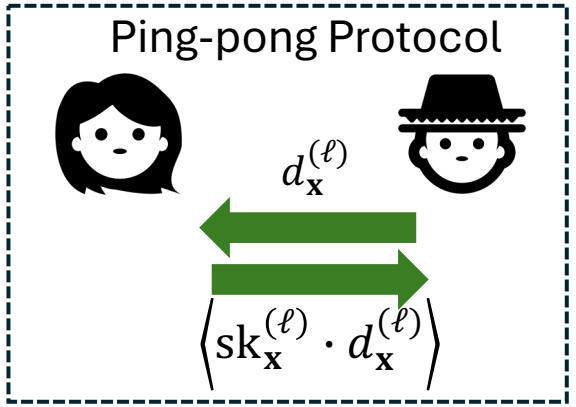
Recursive Construction



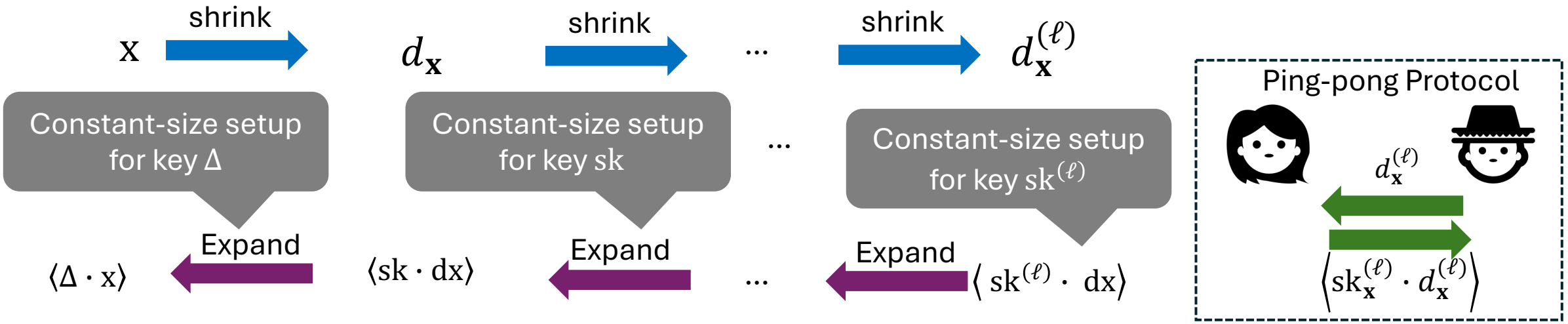


Recursive Construction

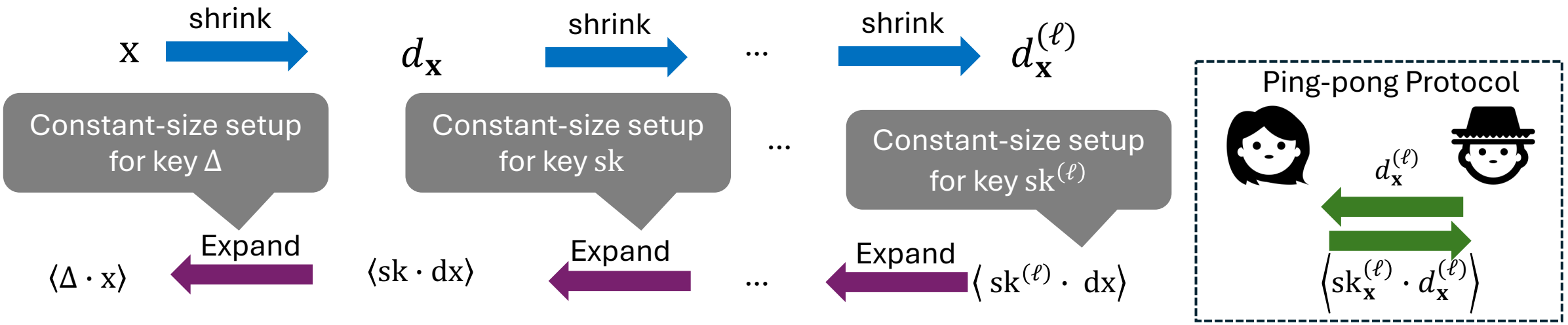




Recursive Construction



Recursive Construction



Observation:
The parties can reuse $\mathrm{sk} = \mathrm{sk}^{(1)} = \cdots = \mathrm{sk}^{(\ell)}$ for inner recursion
⇒ Constant size setup (regardless of $|\mathrm{x}|$)

What's more in the paper

- Subtle Challenges
  - Mismatching the message space $\mathcal{M}$ and key/digest space $\mathcal{C}$
  - Gadget Decomposition for Noise Handling in RLWE-based Construction
- Optimizations:
  - Gold-Seed Grinding:
    - Reducing computation/communication cost from $O(\lambda^3) \rightarrow O(\lambda \log^2(|x|))$
  - One-Digit Gadget Truncation
    - Reduce the computation and communication by $4\times$

Conclusion

- We propose a succinct and concrete efficient CI-VOLE
  - Polylog-size setup and communication
- Applications:
  - Prover-Efficient Zero-Knowledge Proofs
  - Non-interactive Public-key Zero-Knowledge Proof
  - Garbled Input Label Transfer
  - Can you think of more?

Our Non-interactive Mode

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$

$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← d'

$d_x = LacE.Digest(\mathbf{x})$

$d' = Randomize(d_x)$

$sk_k \leftarrow_\$ \mathcal{C}$

$ct_k = H(seed)$

$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

$sk'_x = sk_r \cdot d' + st_k$

sk'_x →

$sk_x = Derandom(sk'_x)$

$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

Our Non-interactive Mode

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$

$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← d'

$d_x = LacE.Digest(\mathbf{x})$

$d' = Randomize(d_x)$

$sk_k \leftarrow_\$ \mathcal{C}$

$ct_k = H(seed)$

$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

$sk'_x = sk_r \cdot d' + st_k$

sk'_x →

$sk_x = Derandom(sk'_x)$

$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

Observe:

sk_k and sk'_x are random

Our Non-interactive Mode

Sender 

 Receiver

ct_{lac} = LacE.Enc((Δ, . . . , Δ); **r**)
ct_r = LHE.Enc(sk_r, **r**)

ct_{lac}, ct_r

(One-time) Setup

Online

d’

d_x = LacE.Digest(**x**)

d’= Randomize(d_x)

sk_x = Derandom(sk'_x)
ct_k= H(seed)

ct_k= H(seed)
k = LHE.Dec(ct_k,sk_k)

Δ · x + **k** = LHE.Dec(sk_x, ct_r · d_x + ct_k) - LacE.Eval(ct_{lac}, **x**)

Observe:
sk_k and sk'_x are random

Our Non-interactive Mode

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← d'

$d_x = LacE.Digest(\mathbf{x})$
$d' = Randomize(d_x)$

$sk'_x = H(d')$
$st_k = sk'_x - sk_r \cdot d'$
$ct_k = H(seed)$
$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

$sk'_x = H(d')$
$sk_x = Derandom(sk'_x)$
$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

Observe:
sk_k and sk'_x are random

Our Non-interactive Mode

Sender 

 Receiver

$ct_{lac} = LacE.Enc((\Delta, \ldots, \Delta); \mathbf{r})$
$ct_r = LHE.Enc(sk_r, \mathbf{r})$

ct_{lac}, ct_r →

(One-time) Setup

Online

← d'

$d_x = LacE.Digest(\mathbf{x})$
$d' = Randomize(d_x)$

$sk'_x = H(d')$
$st_k = sk'_x - sk_r \cdot d'$
$ct_k = H(seed)$
$\mathbf{k} = LHE.Dec(ct_k, sk_k)$

$sk'_x = H(d')$
$sk_x = Derandom(sk'_x)$
$ct_k = H(seed)$

$\Delta \cdot x + \mathbf{k} = LHE.Dec(sk_x, ct_r \cdot d_x + ct_k) - LacE.Eval(ct_{lac}, \mathbf{x})$

k is no longer pre-computable

Observe:
sk_k and sk'_x are random