Distributed Zero Knowledge & Applications to Secure Computation

Elette Boyle

IDC Herzliya, Israel

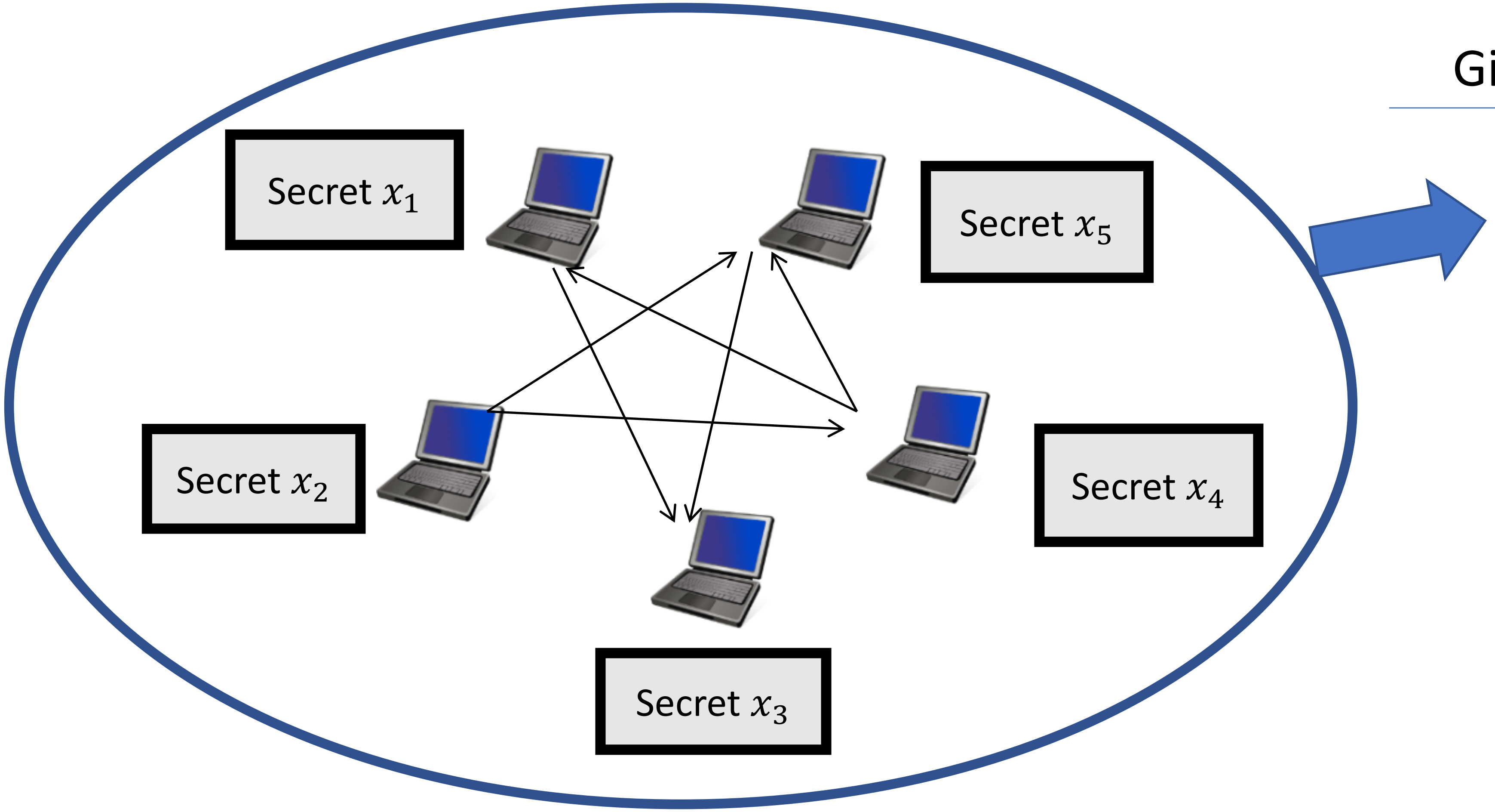
IDC HERZLIYA | Efi Arazi School of Computer Science

Based on joint works with: Dan Boneh, Henry Corrigan-Gibbs, Niv Gilboa, Yuval Ishai, and Ariel Nof

Secure Multi-Party Computation (MPC)

[Yao82, GMW87, BGW88, CCD88]

Jointly compute on secret data, **without revealing the data**



Given public function f :

Learn $f(x_1, x_2, \ldots, x_5)$

And learn *nothing else* about $x_1, \ldots, x_5$

Even given a subset of *adversarial* parties

Classic "Killer App" of Zero Knowledge (ZK)

(Standard) active security

Force passive behavior via
Zero Knowledge [GMW86,GMW87]

(Weak) passive security

Passive-to-Active Compilers: Now

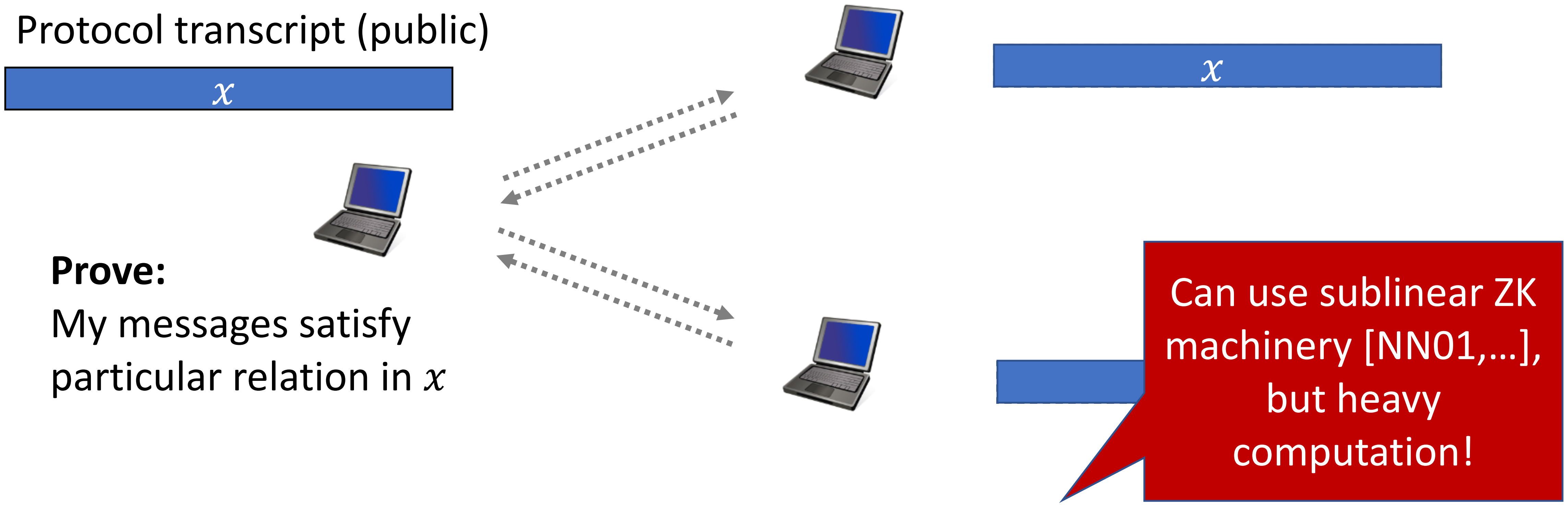
- Several approaches
  - ZK with each message [GMW87]
  - IPS compiler [IPS08]
  - Cut and choose [MF06,LP07]
  - AMD Circuits [GIPST14]
  - Targeted “Authenticated Beaver triple” generation [BDOZ11,DPSZ12,…]
  - …

- Still important area of research – working to minimize induced overhead

(Standard) active security

(Weak) passive security

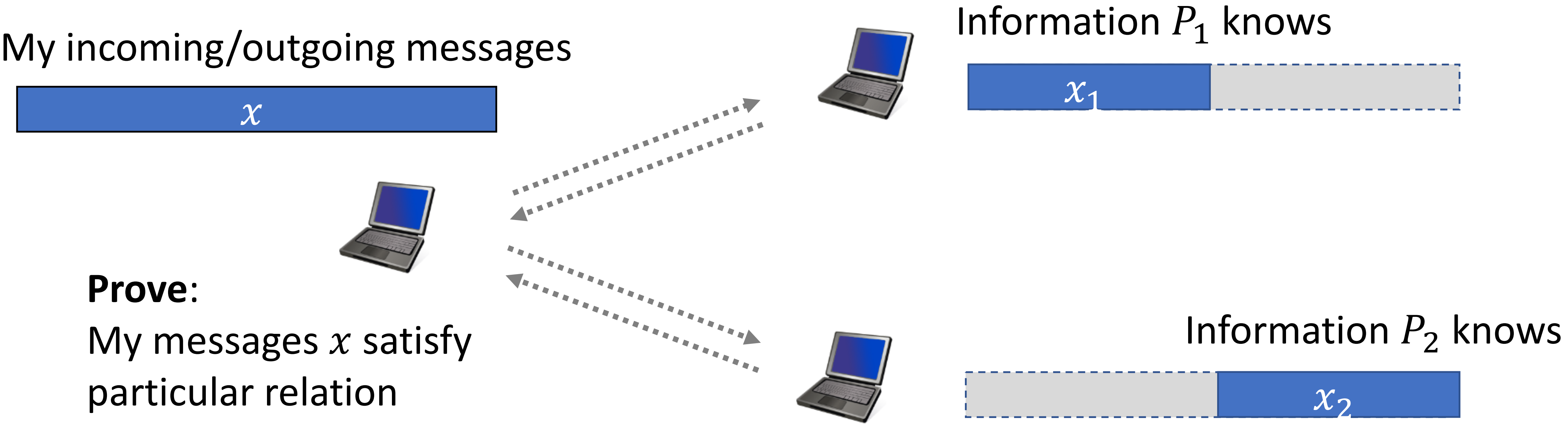
Original GMW Compiler [GMW87]



Can only use on information **everybody knows**

Requires **overhead** in communicating, **more complex** statements

Our Motivation: A Lighter Approach

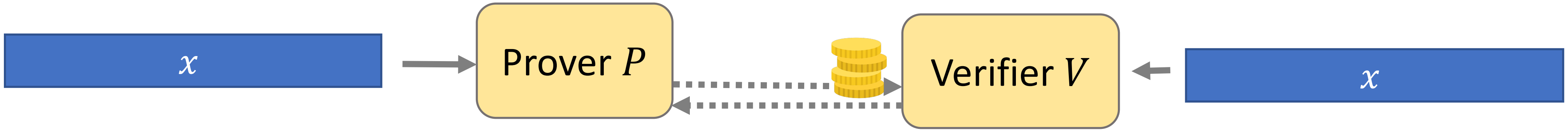


Prove:
My messages x satisfy particular relation

Simple relations, minimal additional communication

Wanted: Method to **prove statements held distributedly across verifiers**

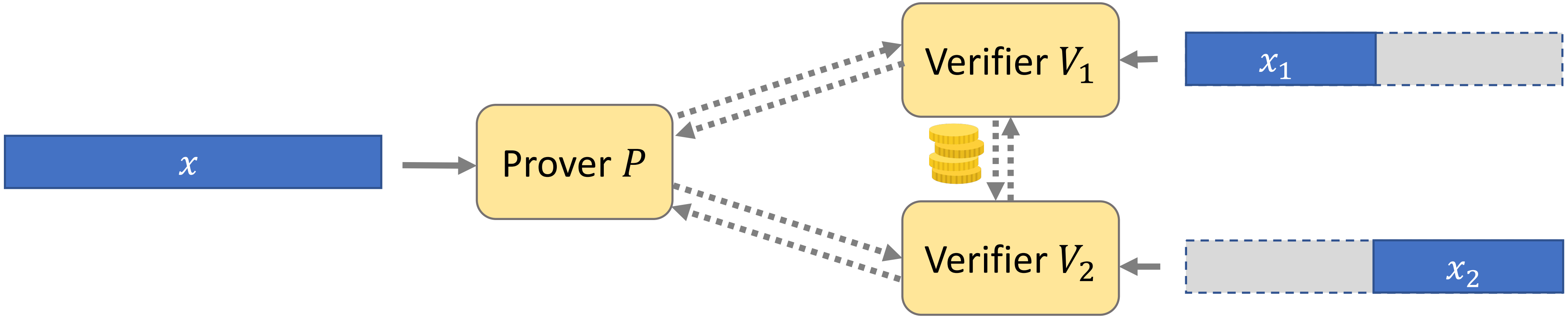
Zero Knowledge [GMR85]



Zero Knowledge:
Verifier learns nothing beyond
$(x,\ x \in L)$

Distributed -Verifier Zero Knowledge [BBCGI19]

- Zero-Knowledge Proofs on (secret-shared / committed / distributed) data



Zero Knowledge:
V_i learns nothing beyond
$(x_i,\ x \in L)$

New motivated settings & metrics

- No verifier knows x $\Rightarrow$ nontrivial even for simple languages L

- New applications with different complexity goals
  - Care about <u>proof size</u>

Similar-Sounding (But Different) Notions

- "DIZK: Distributed Zero Knowledge" (& related) [WZCPS18,EFKP20]
  - Distributes proof generation across cluster / parallel threads, for efficiency
- Interactive Distributed Proofs [KOS18,CFP19,NPY20]
  - Verifier = nodes of a communication graph
  - Prove properties of the graph

Today: Brief Survey of D-ZK

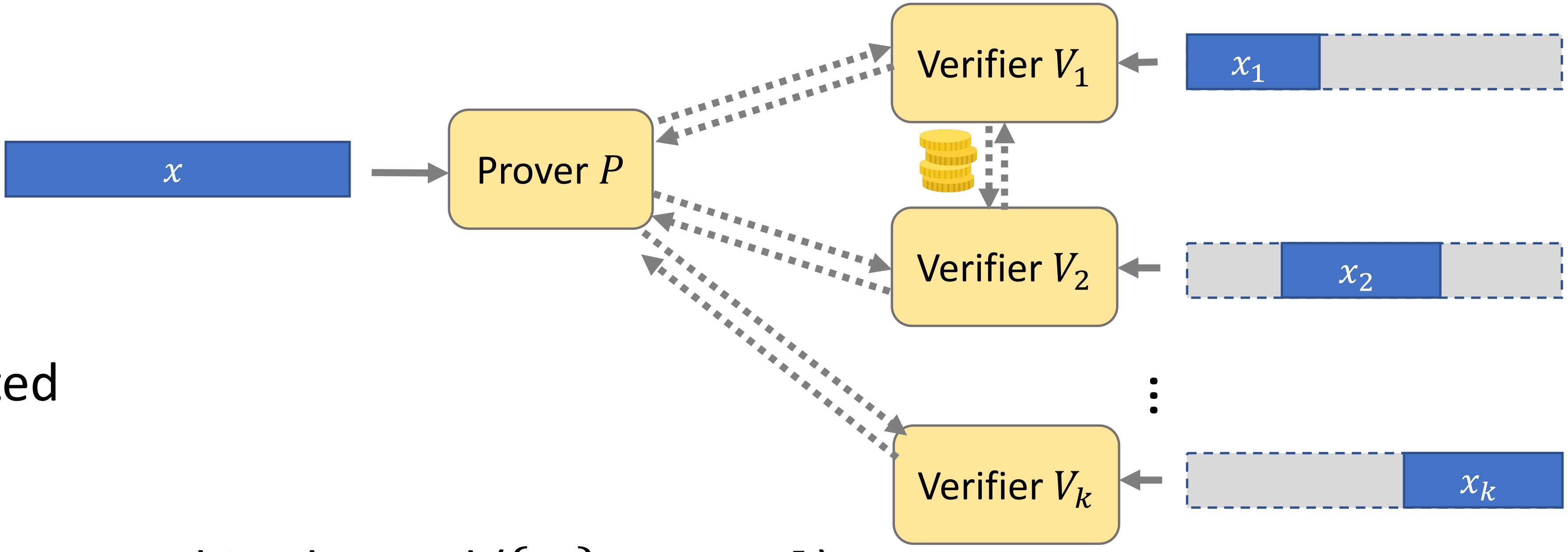
- Few More Words on Definition
- Construction Approach
  - via “**Fully Linear** PCP/IOP”
- Applications to Secure Computation

Punchlines:

Low-cost D-ZK protocols
for simple languages

Compilers to malicious security with
sublinear communication overhead

Distributed-Verifier Zero Knowledge [BBCGI19]



- Completeness
  - Honest P accepted
- Zero Knowledge
  - Corrupt $\{V_i\}_{i \in S}$ learn nothing beyond $(\{x_i\}_{i \in S}, x \in L)$
- Soundness
  - Corrupt P (no V_i): Honest V_i accepts $\Rightarrow x \in L$
  - Corrupt $P + V_i$'s : More subtle… need robustness of x across Verifiers

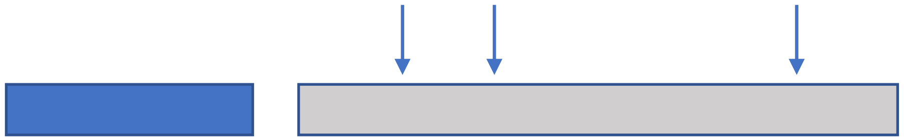
Constructions:

A Jaunt Through “Fully Linear PCP/IOPs”

Core Tool: "Fully Linear PCP"

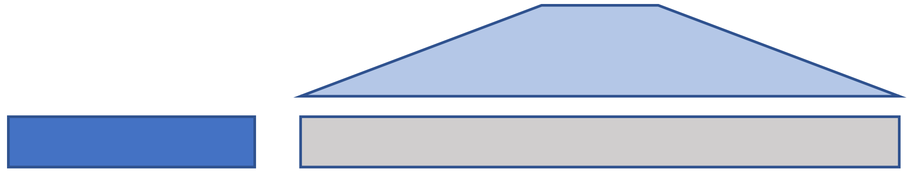
Probabilistically Checkable Proof (PCP)

[AS90,ALMSS91]



Statement x Proof π

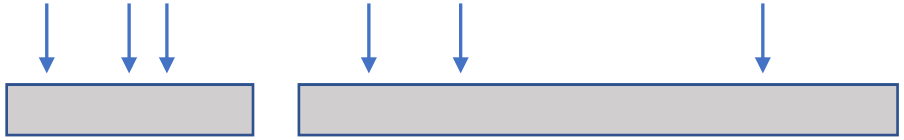
Linear PCP [IKO07,BCIOP13]



Statement x Proof π

PCP of Proximity [BGHSV14,DR14]

Verify: x is *close* to L

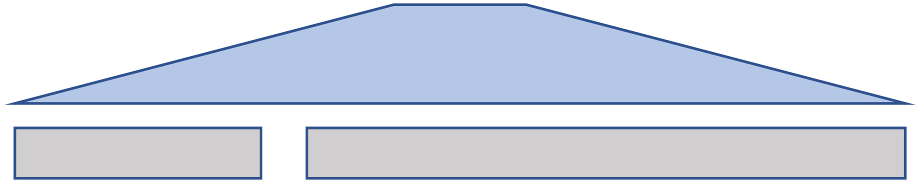


Statement x Proof π

zk-**Fully Linear** PCP [BBCGI'19]

V learns <u>only</u> $x \in L$

Verify: $x \in L$

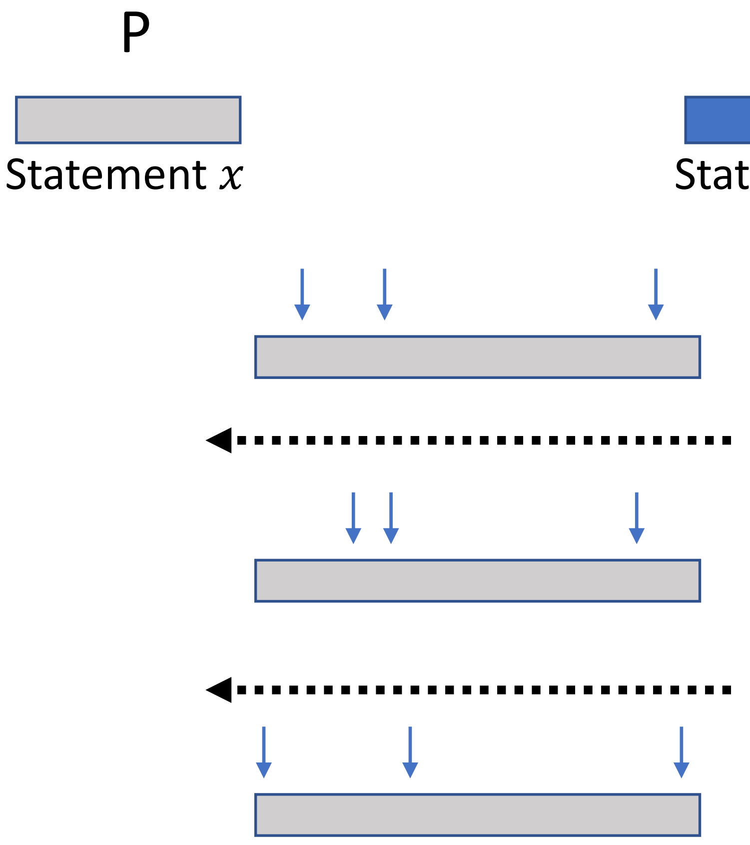


Statement x Proof π

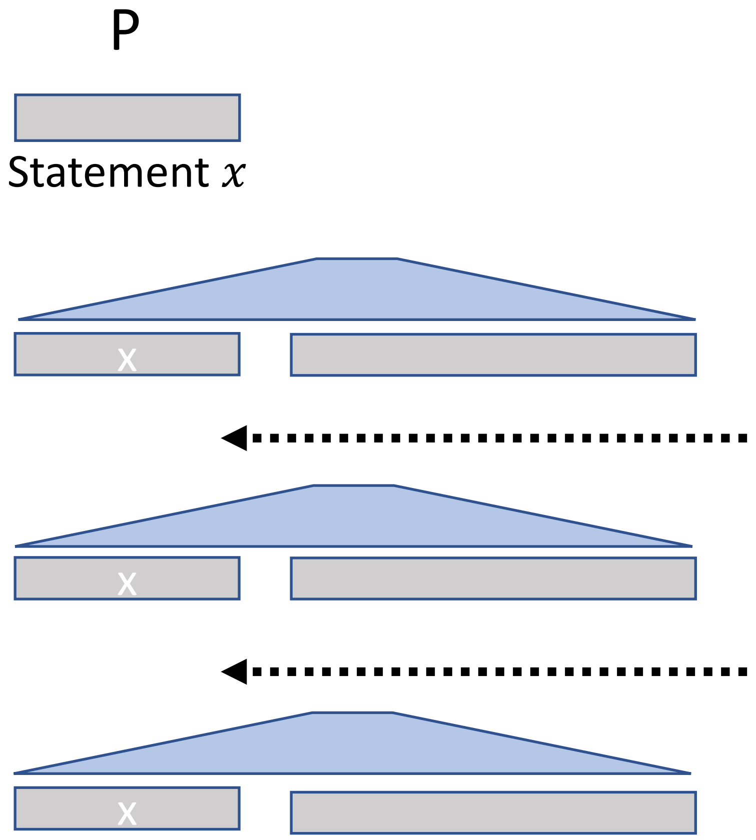
Both cases:
Can use Fiat-Shamir to collapse to 1 round

Core Tool: “Fully Linear **IOP**”

Interactive Oracle Proof (IOP) [BCS16,RRR16]

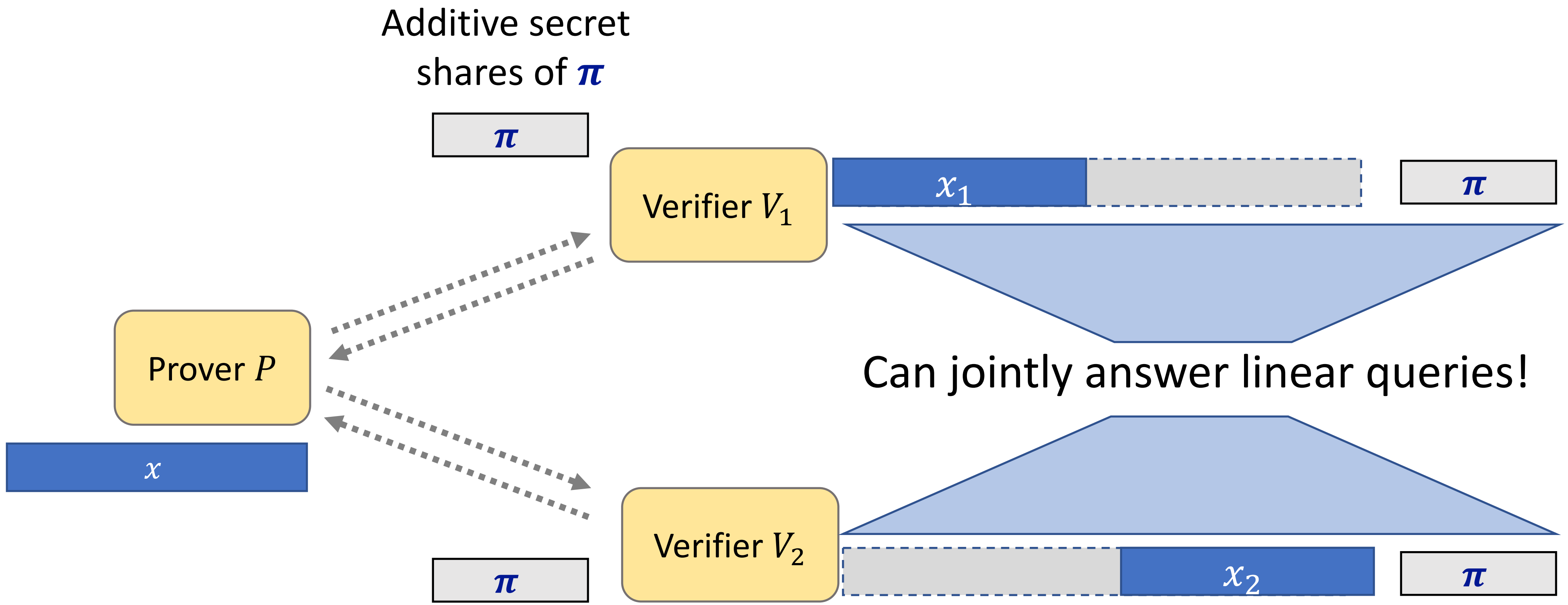


zk-**Fully Linear** IOP [BBCGI’19]



V learns <u>only</u> $x \in L$

Distributed-ZK from zk-Fully Linear PCP/IOP



Building zk-Fully Linear PCP/IOP

Many FL-PCP implicit in literature (eg [GKR08,CMT12,GGPR13,RRR16,…])

Our setting (ZK + short proofs): Builds on Ideas of Sum Check

Theorem [BBGCI19]: For $L \subseteq \mathbb{F}^n$

- Degree 2 over $\mathbb{F}$: **FL-PCP** of proof size $\tilde{O}(\sqrt{n})$, error $\mathbb{F}^{-1}$
- Degree $O(1)$ over $\mathbb{F}$: **FL-IOP** of proof size $\tilde{O}(\log n)$, rounds $\log n$, error $\mathbb{F}^{-1}$

Remarks:

- No computational assumptions!
- Extends simply to small fields, rings via extension

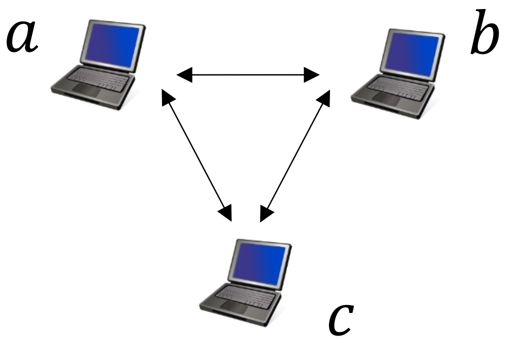
Applications:

Back to Multi-Party Computation (MPC)

Secure Multi-Party Computation (MPC)

What is the **communication** complexity of **securely** evaluating f?

- **In theory**: $\tilde{O}(|\text{inputs}| + |\text{outputs}|)$ [G09,BGI16a]
  - Based on heavy cryptographic tools (FHE, HSS, …)

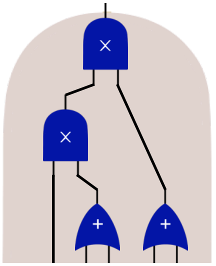


(Black-box use of PRG)

- **In practice**: $(\alpha \cdot |C|)$ elements/party, small const $\alpha \geq 2$
  - Long line of work improving α in various settings

$\boldsymbol{\alpha \geq 2}$ Step 2: (standard) “active” security

Lightweight $\boldsymbol{\alpha = 1}$ Step 1: (weak) “passive” security



C = (Boolean/arithmetic) circuit representation of f

The Goal: Sublinear Additive Overhead

Same (amortized)
$\boldsymbol{\alpha}$ elemts/party/gate

(Standard) active security

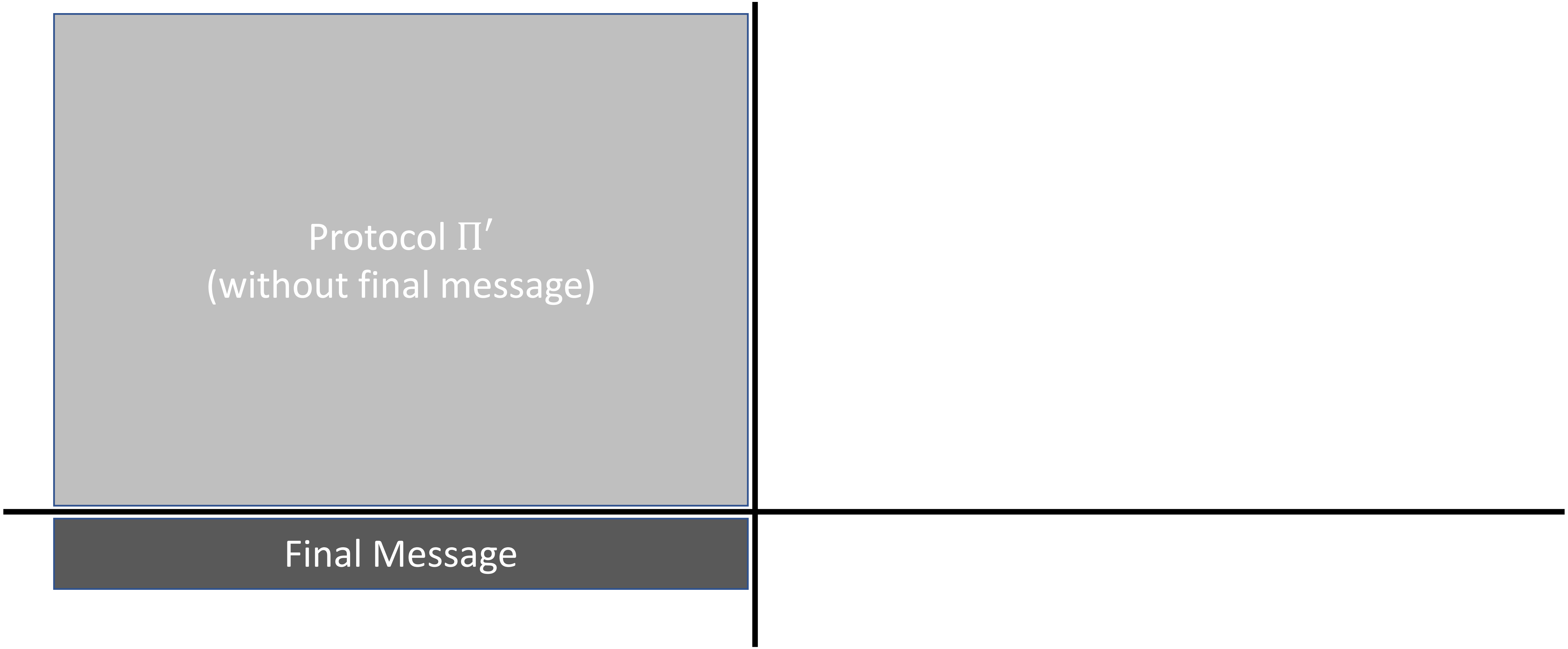
+ o(|C|) communication

$\boldsymbol{\alpha}$ elemts/party/gate

(Weak) passive security

High-Level Plan

Start with passive-secure protocol with useful “natural” structure



High-Level Plan

Start with passive-secure protocol with useful “natural” structure

Protocol Π'
(without final message)

Final Message

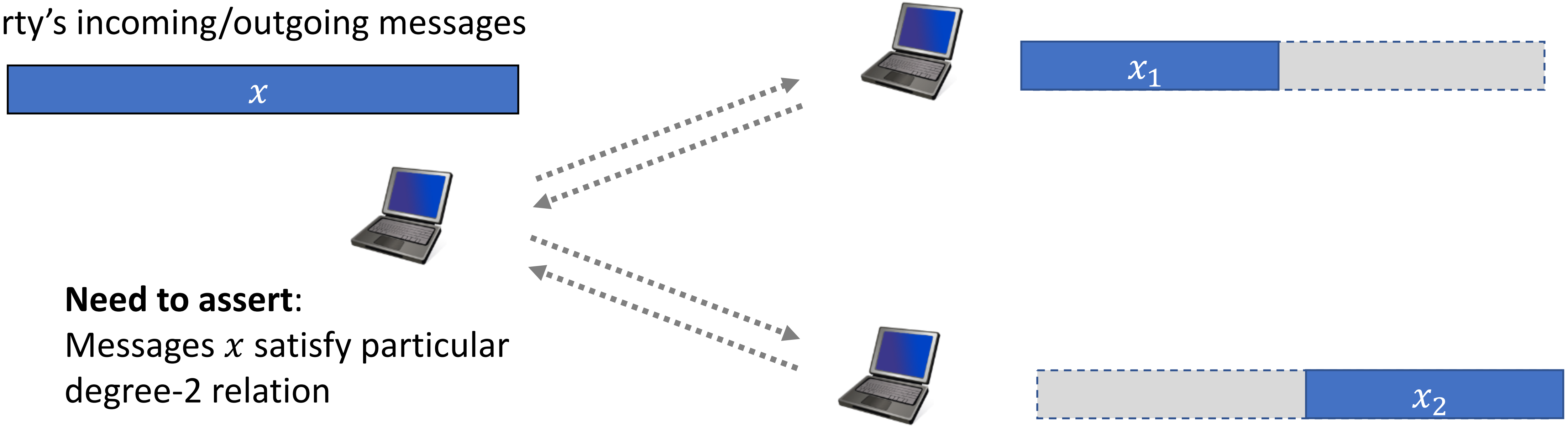
Somehow prove in ZK that Π' was performed correctly

If any party rejected: ABORT

D-ZK Compiler - Type 1 [BBCGI19,BGIN19,BGIN20]

Each party proves its messages in Π' were computed correctly

Party's incoming/outgoing messages

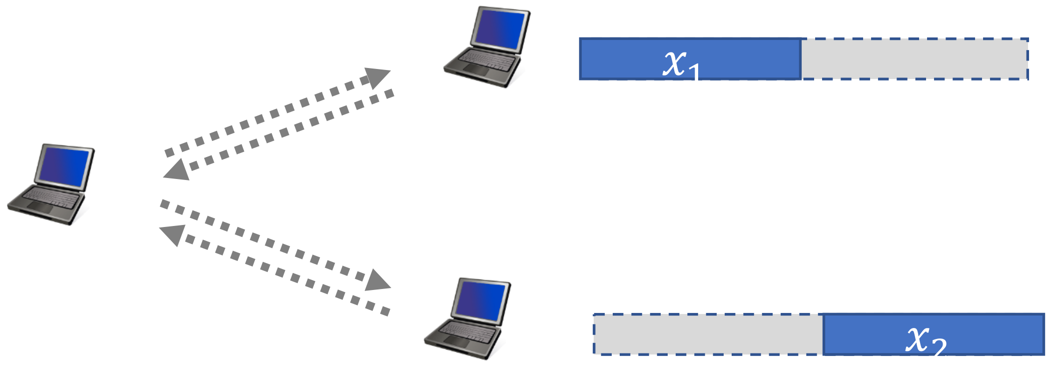


Need to assert:
Messages x satisfy particular degree-2 relation

- Degree 2 = "simple" relation: We have sublinear-size proofs!
- Soundness ~1/(field size). But: amplify by embedding to larger space (still o(|C|))

D-ZK Compiler - Type 1 [BBCGI19,BGIN19,BGIN20]

- Works great for **3 parties, 1 corruption**
  - **Either** prover **or** verifier corrupt



- Doesn't work directly for $\boldsymbol{n}$ **parties**, $\boldsymbol{t}$ **corruptions,** $n = 2t + 1$
  - Challenges with colluding prover & verifiers…

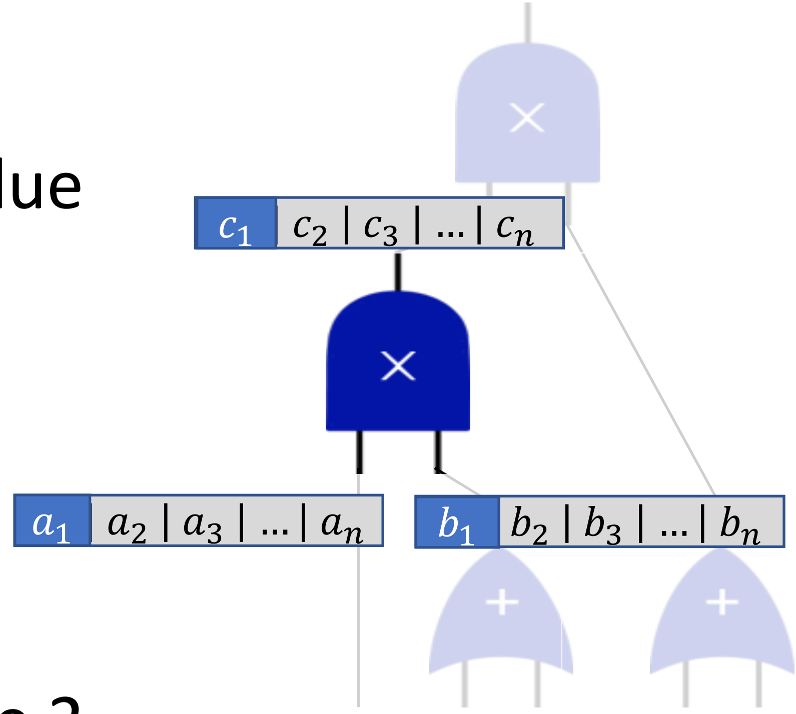
D-ZK Compiler - Type 2 [BBCGI19]

Parties jointly prove secret shared wire value are consistent

- Statement: Robust secret shares of each wire value

- Nobody knows the full statement!
- Parties **jointly emulate** Prover

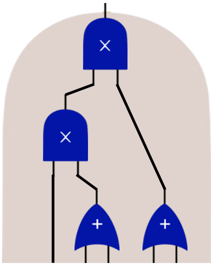
- Prover of degree 2 relation computable in degree 2
  $\Rightarrow$ can compute shares of proof π non-interactively



Honest Majority MPC

- **3 parties, 1 corruption** (“3PC”)
  - Motivated setting: “Minimal” across MPC settings
    eg: [MRZ15,AFLN+16,ABFL+17,LN17,FLNW17,CGHI+18,GR018,NV18,EnOP+19]
  - Comparison:

Over large field:	$\boldsymbol{\alpha = 2}$	[CGHIKLN18, NV18]
Over Boolean:	$\boldsymbol{\alpha = 7}$	[ABFLLNOWW17]
Any field or $\mathbb{Z}_N$	$\boldsymbol{\alpha = 1}$	[BBCGI19,BGIN19]
		Compiling eg, [AFLNO16,KKV18]



- **$\boldsymbol{n}$ parties, $\boldsymbol{t}$ corruptions,** $n = 2t + 1$

Over large field:	$\boldsymbol{\alpha = 3}$	[CGHIKLN18]
Over Boolean/$\mathbb{Z}_{2^k}$:	$\boldsymbol{\alpha > 40}$	[CGHIKLN18]
Any field or $\mathbb{Z}_N$	$\boldsymbol{\alpha = 3t/(2t+1)} \leq 1.5$	[BBCGI19,GL20,BGIN20]
		Compiling [DN07]

*Dis*honest Majority MPC…?

Wait a second.

We expressly needed **robust** sharings across parties…

not possible with dishonest majority…

D-ZK Compiler - Type 3

New!

[BGIN??]

- Without an honest majority!
- MPC in Preprocessing Model
  - “Dealer” in preprocessing phase (emulated by all parties)

Preprocessing Phase (input independent)

Cheap online phase

Our idea: “Dealer” acts as an extra verifier **guaranteed to be honest**

MPC with Preprocessing

Preprocessing Phase
(input independent)

Cheap online phase

- Relevant metrics: (1) Size of preprocessing data
  (2) Online communication
- Prior solutions: Either (1) or (2) had at least linear $\Omega(|C|)$ overhead
  [BDOZ11,NNOB12,DPSD12,DZ13,CDE+18,CG20,…]
- Our solution: **Both (1) + (2) have sublinear $\mathbf{o}(|\mathbf{C}|)$ overhead**!

In Summary

- Distributed-Verifier ZK
- Constructions
  - Via Fully Linear PCPs / IOPs
  - Short (sublinear) proofs for simple languages!
  - No computational assumptions, over $\mathbb{F}$ & rings $\mathbb{Z}_N$
- Applications to MPC
  - Sublinear communication overhead (passive -> active)

Open Questions

- Fully Linear PCP/IOP:
  - More efficient constructions for specific simple languages
- Alternative passive-to-active compilers
  - Better efficiency
  - Garbled circuit based passive protocols?
  - Other passive protocol structures, eg for mixed-mode computations?