The MPC-in-the-head paradigm

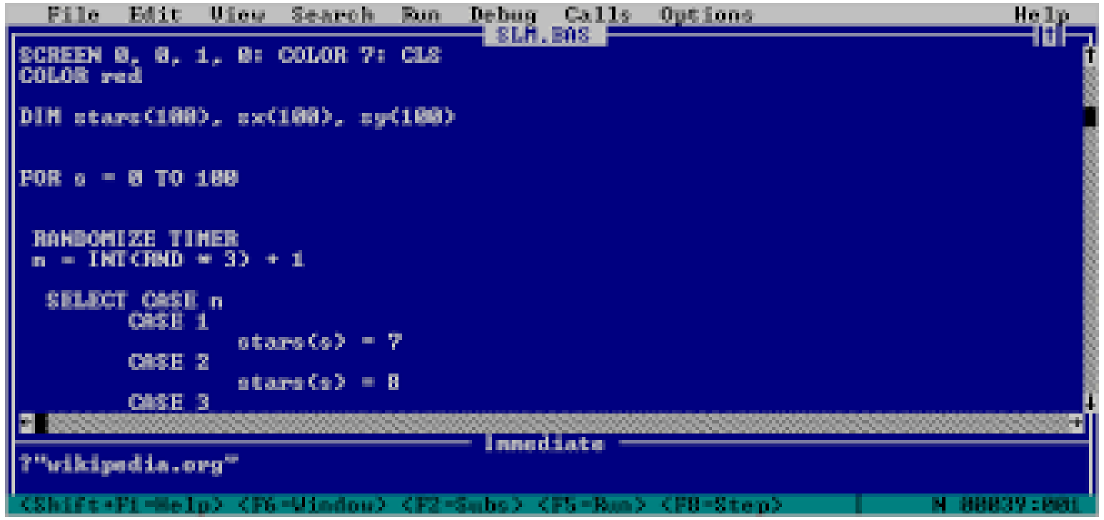
Peter Scholl

Carsten Baum

Plan for today

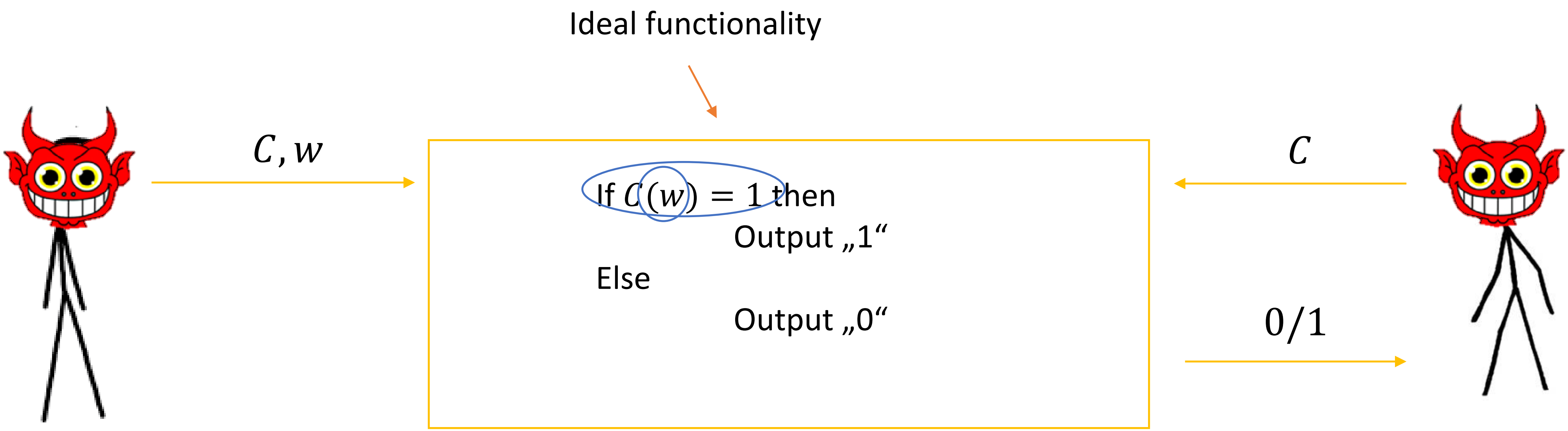
1. Basics of MPC-in-the-head (now)
2. The Ligero proof system & VOLEs
3. VOLE-in-the-head and FAEST

What we will cover in Session 1



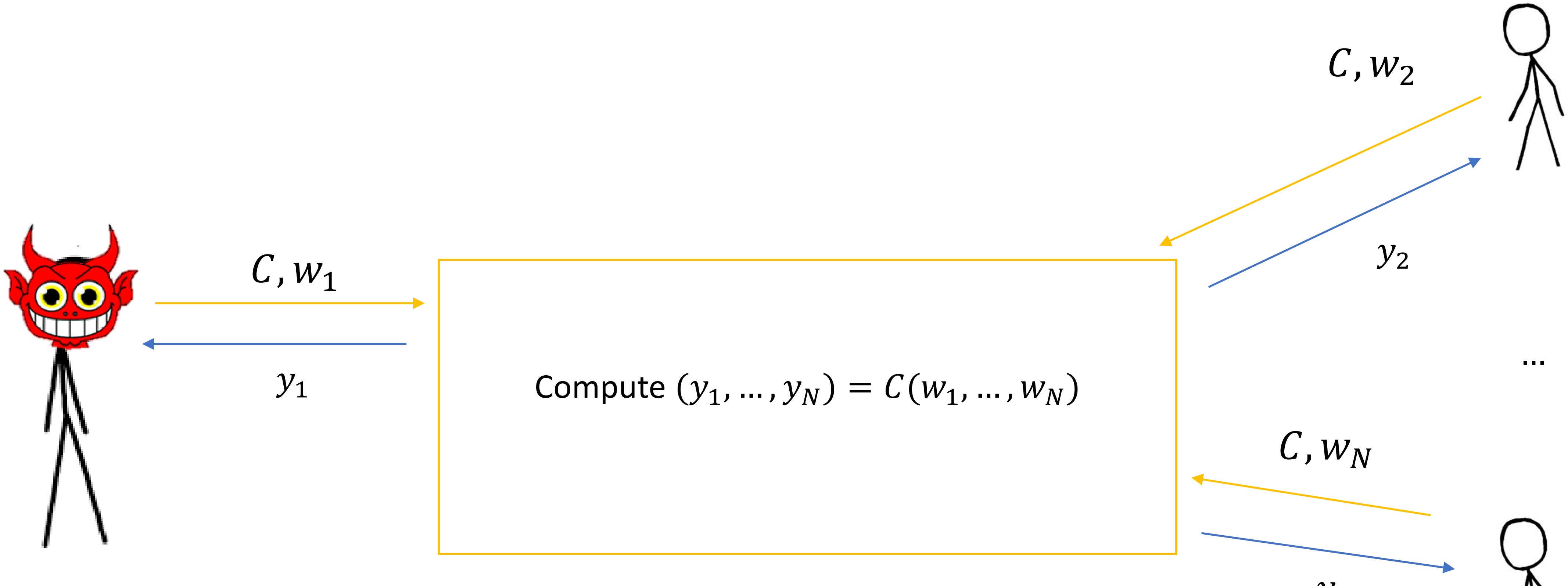
1. What is MPC?
2. From MPC to MPC-in-the-head
3. The KKW construction

Zero-Knowledge Proofs



1. Completeness
2. Knowledge Soundness
3. Zero-Knowledge

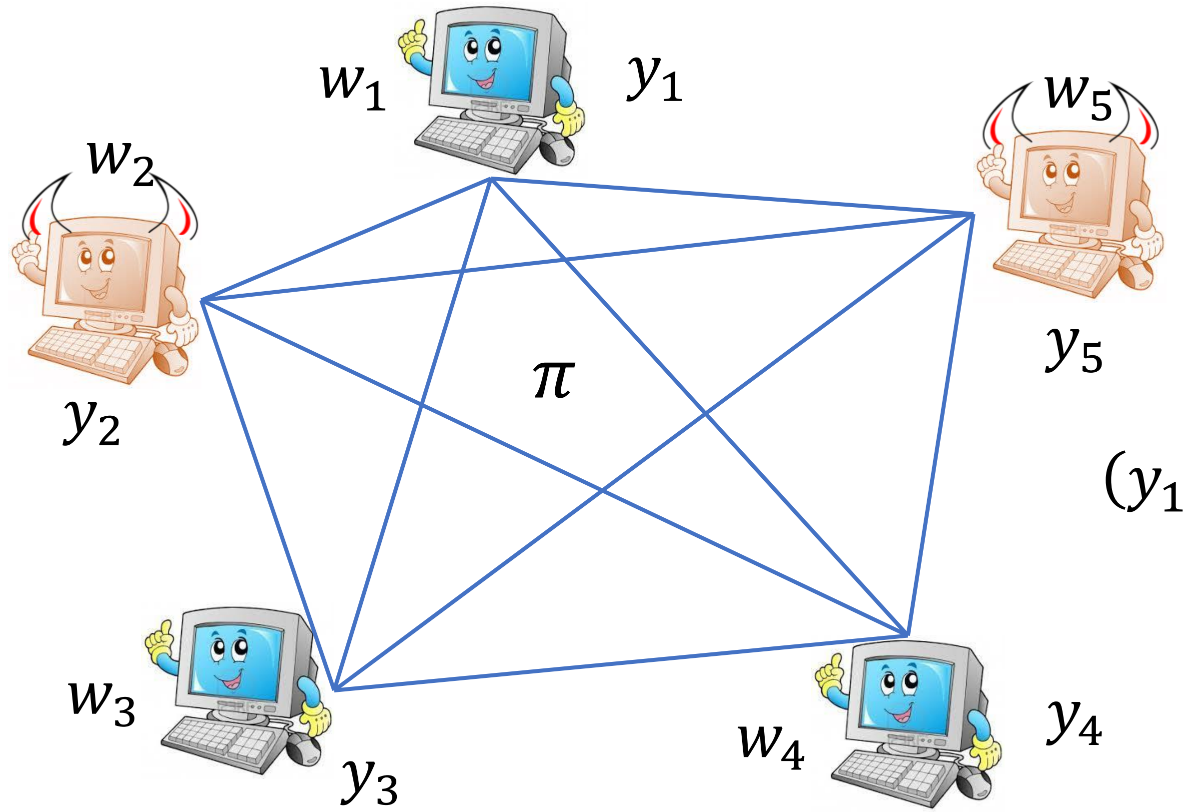
Multiparty Computation



1. Correctness
2. Privacy

ZK is restricted form of MPC

Multiparty Computation (MPC)



$(y_1, \ldots, y_5) = C(w_1, \ldots, w_5)$

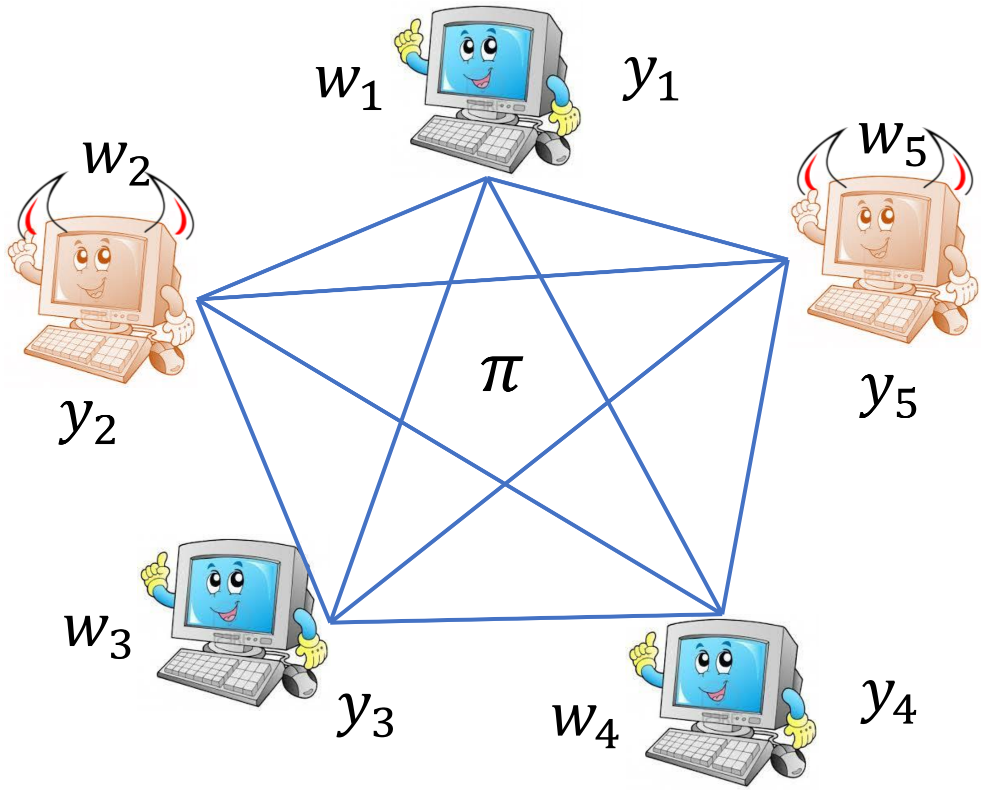
Correctness: if parties learn the output, then it is y_i

t_p-Privacy: no t_p parties can learn anything beyond their inputs and outputs from π

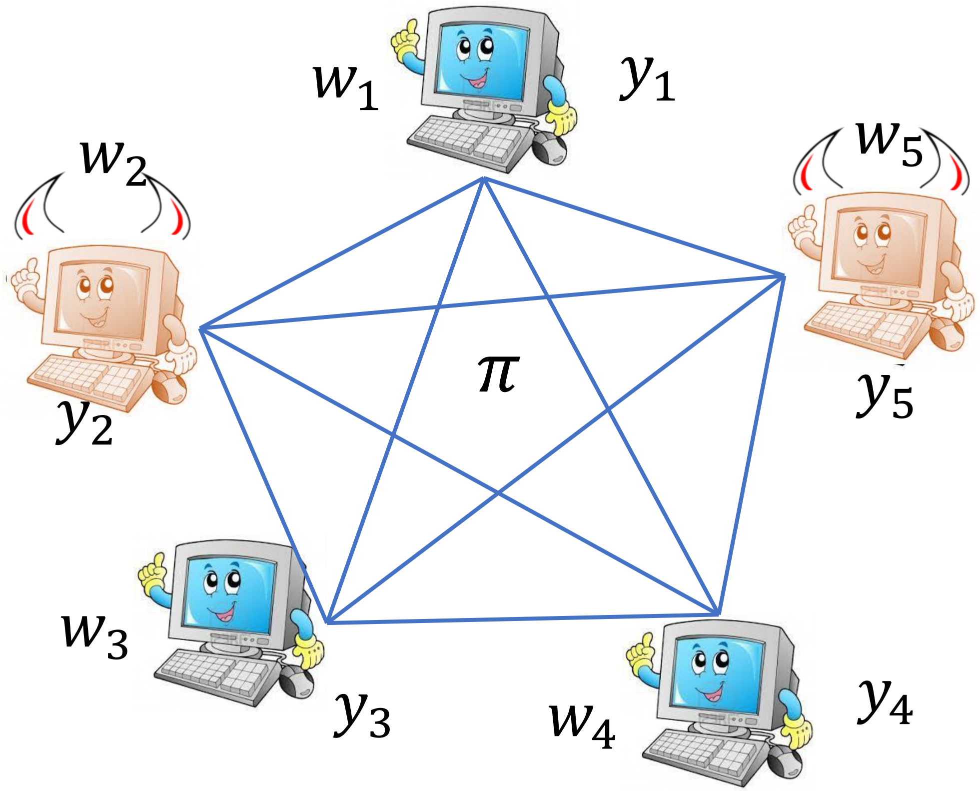
t_r-Robustness: If $\leq t_r$ parties are actively corrupt, then honest parties output y_i or $\perp$

Static vs. adaptive corruptions

Static



Adaptive



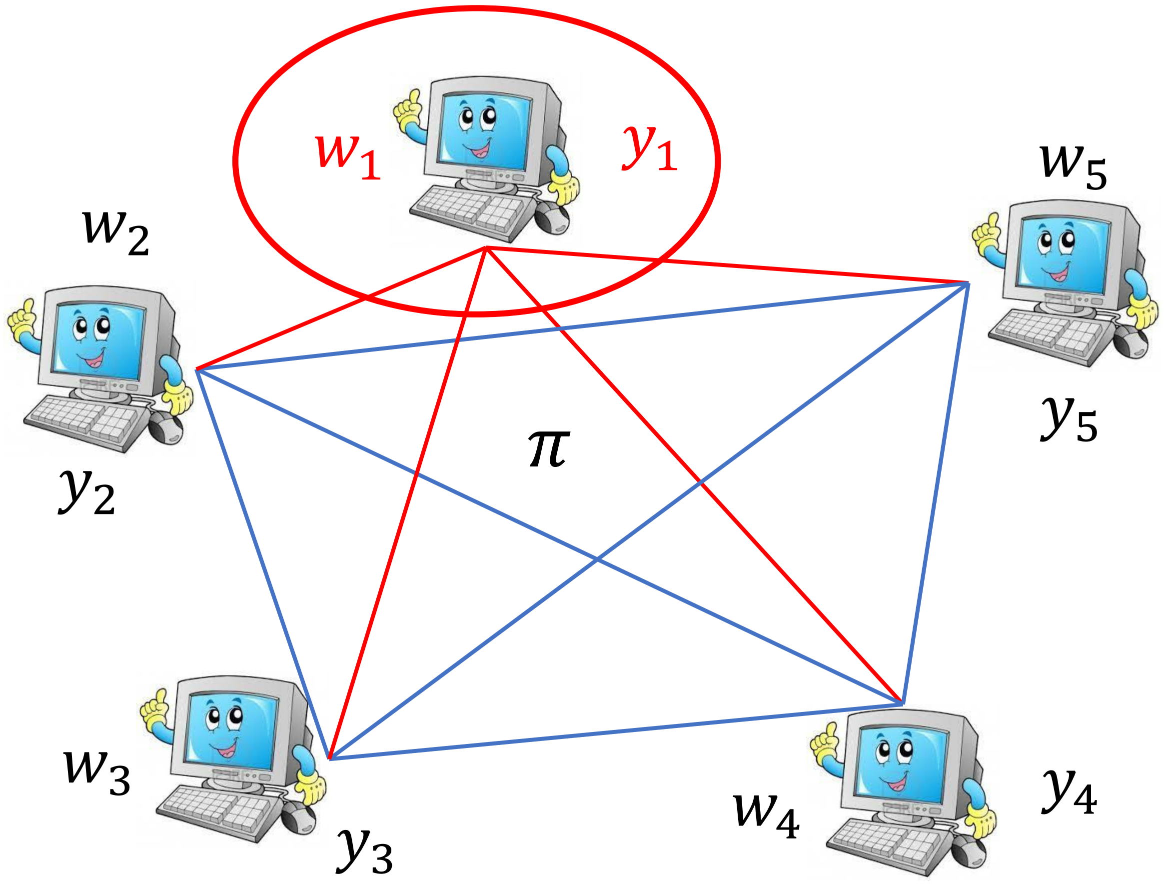
Views

View of P_1

1. All inputs of P_1
2. All outputs of P_1
3. All messages P_1 sent
4. All messages P_1 received

View of adversary

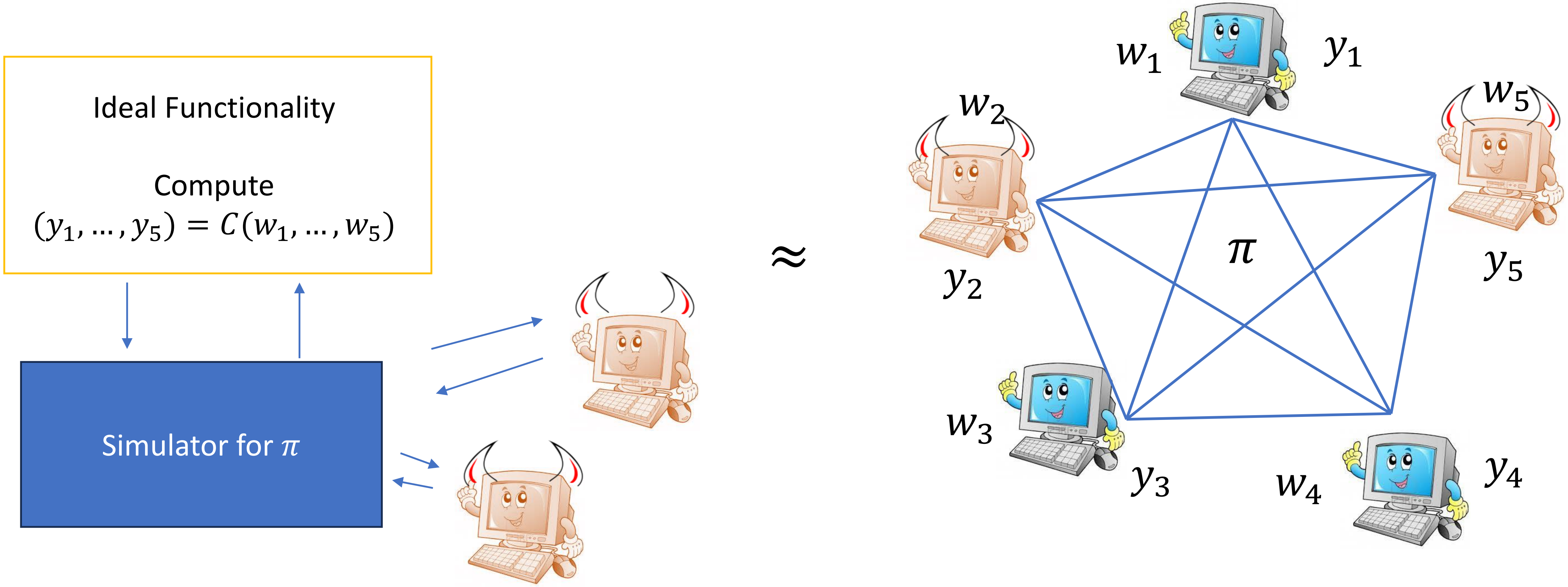
Views of all *corrupt* parties



Security – the simulation paradigm

Ideal World

Real world



Security - Formally

Let A be a PPT algorithm called *adversary.*

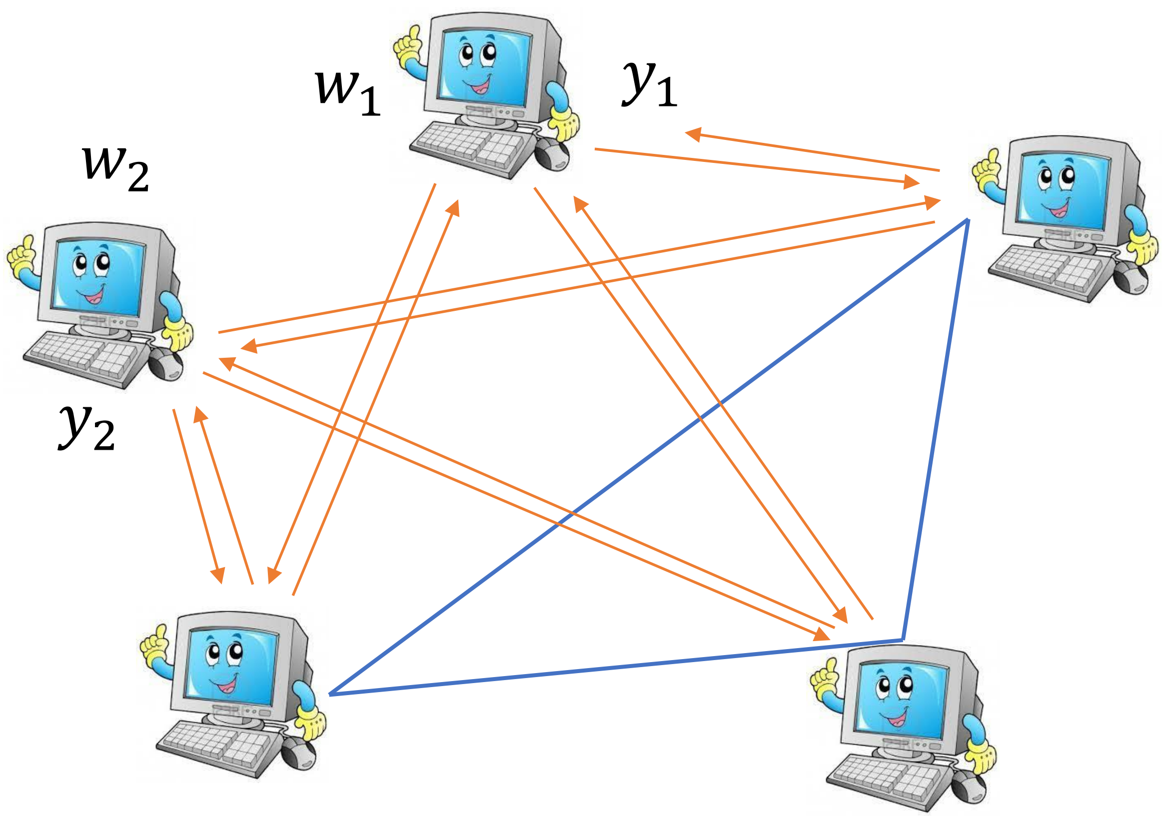
Let $view_{\pi,t}((x_i)_{i\in[N]}, P_1, \dots, P_N, A)$ be the distribution of the protocol messages where A can corrupt at most t parties.

t_p or t_r depending on setting

Let $S(A, F(C, (x_i)_{i\in\bar{I}}))$ be the distribution of messages generated by S interacting with A corrupting parties in $I, |I| \leq t$ as well as F.

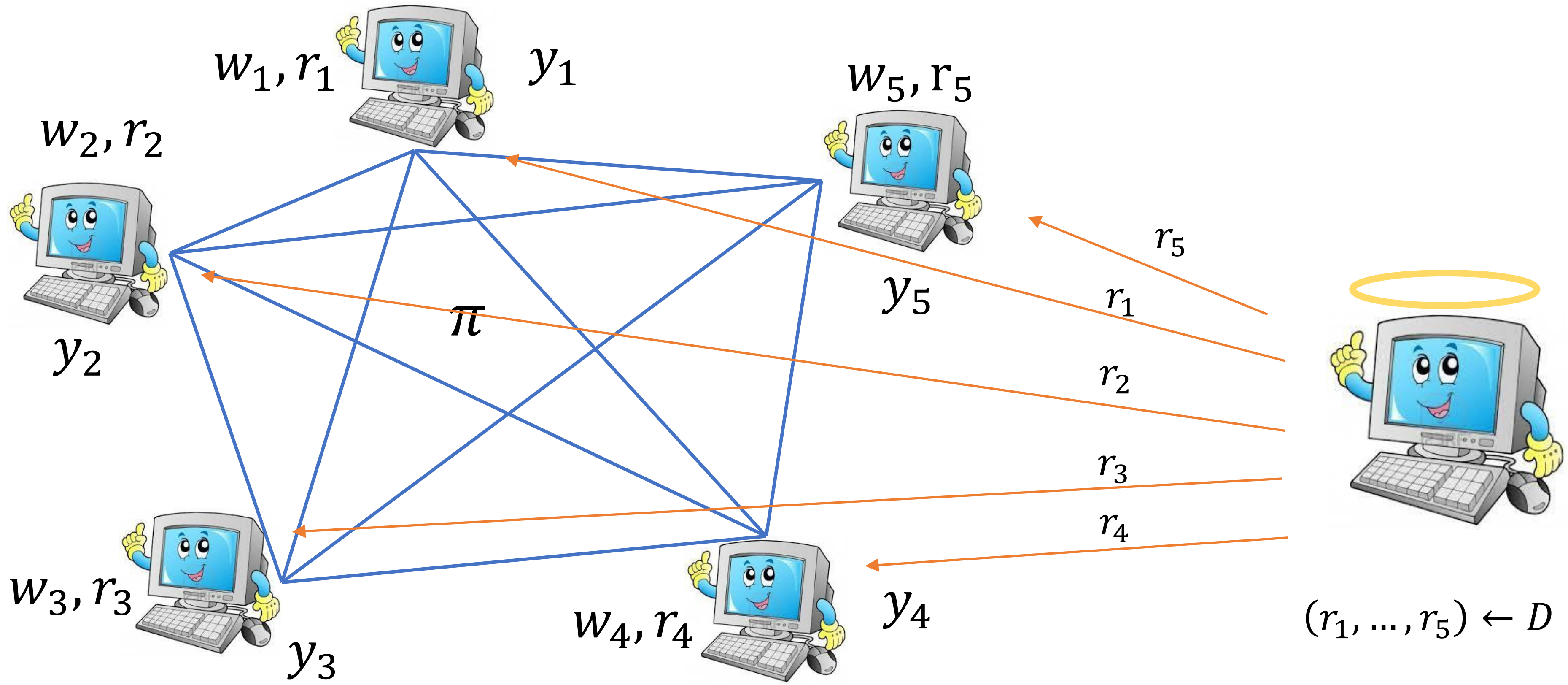
Then π is secure if $view_{\pi,t} \approx S(A, F(C, (x_i)_{i\in\bar{I}}))$ for all $x_1, \dots, x_N$ and C.

Client-Server MPC



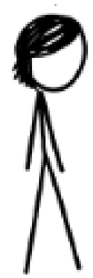
$$(y_1, y_2) = C(w_1, w_2)$$

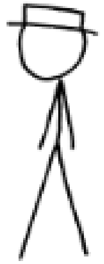
MPC in the preprocessing model



Examples of correlated randomness
- Secret sharing of multiplication triples or bits
- Public key and secret sharing of decryption key

Commitments [Blu82]





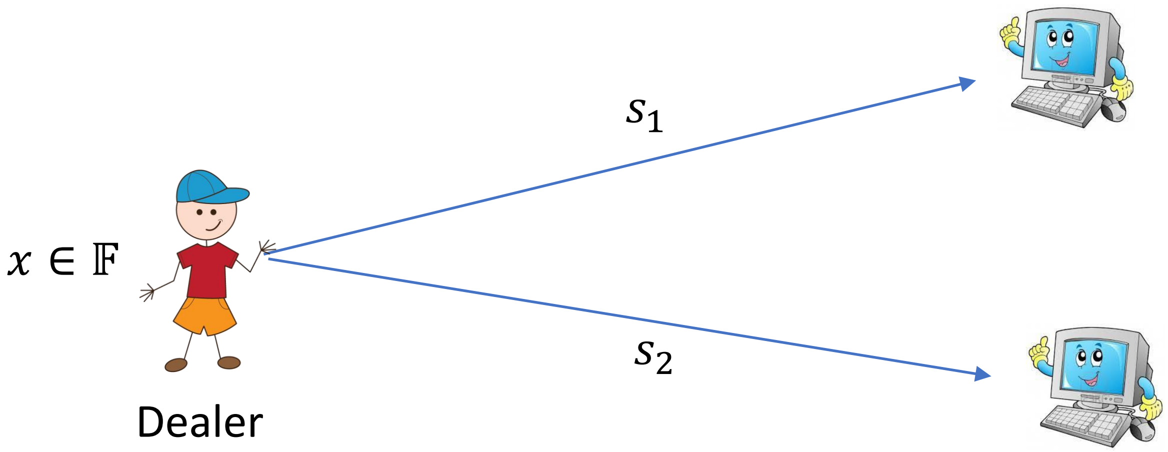
Commitments:

- $Com_{ck}(x, r) \rightarrow c$
- $Open_{ck}(x, r, c) \rightarrow \{\bot, \top\}$

Properties:

1. Binding: can use $Open_{ck}(\cdot,\cdot, c)$ only with (x, r)
2. Hiding: $\{Com_{ck}(x,\cdot)\} \approx \{Com_{ck}(0,\cdot)\}$
3. Equivocable: ck can be generated such that $Open_{ck}(\cdot,\cdot, c)$ works for other x'

Secret Sharing

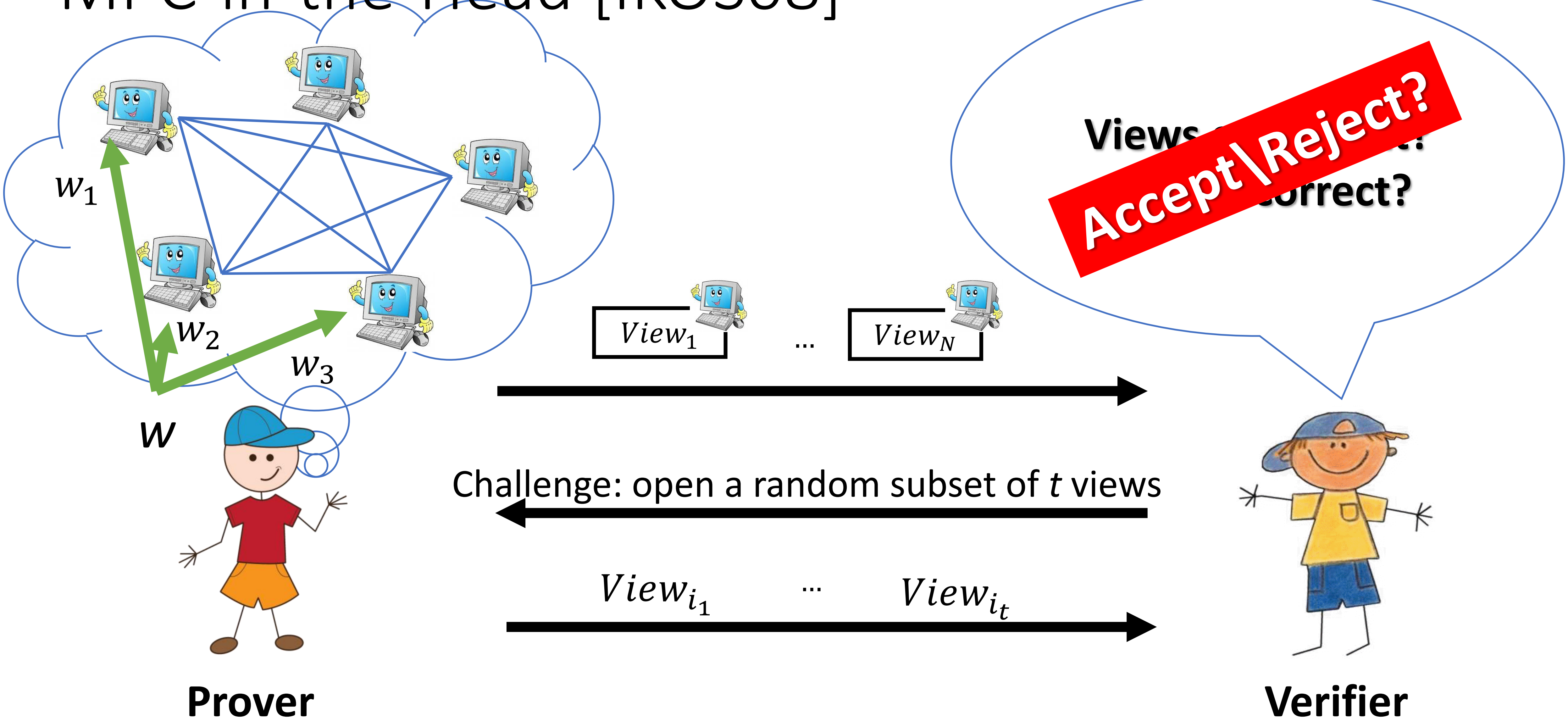


$$(s_1, \ldots, s_n) \leftarrow Share(x)$$
$$y \leftarrow Reconstruct(s_1, \ldots, s_t), y \in \mathbb{F} \cup \{\perp\}$$

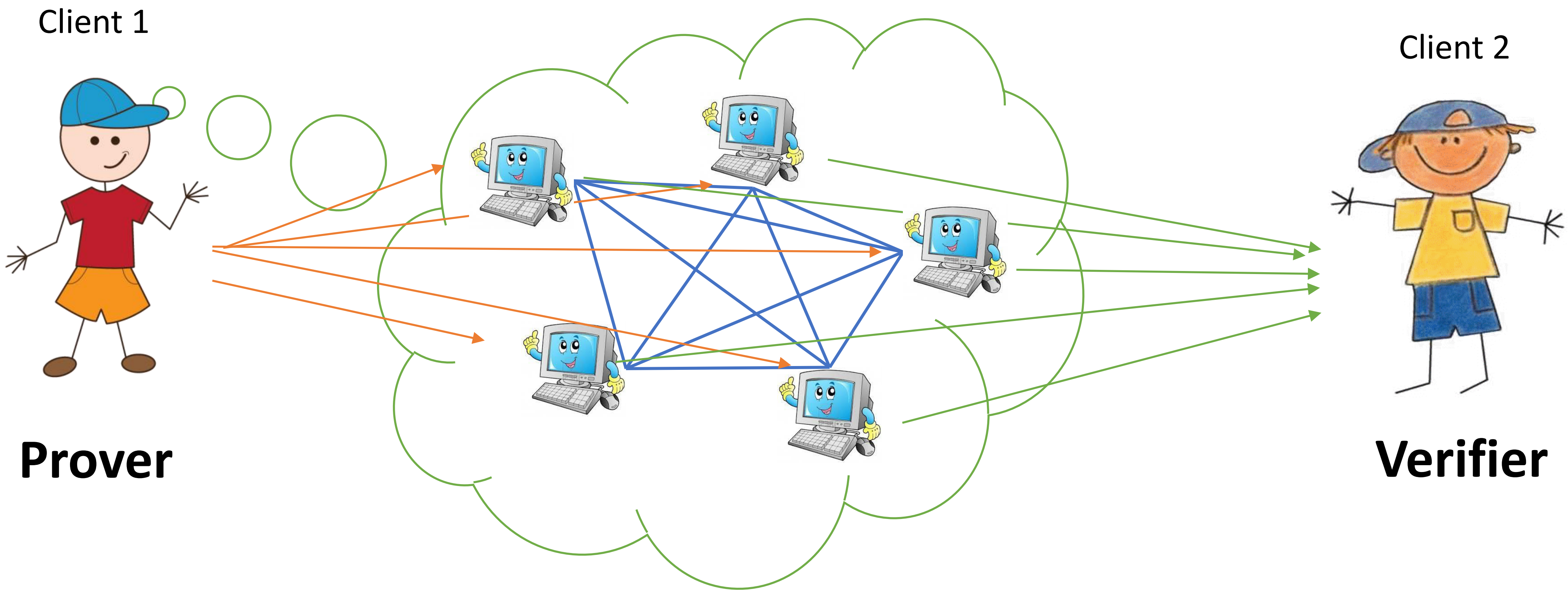
t-privacy: any set of t shares reveals no information about x
$t+1$-reconstruction: any set of $t+1$ shares allows reconstruction of x

} t_p privacy of MPC

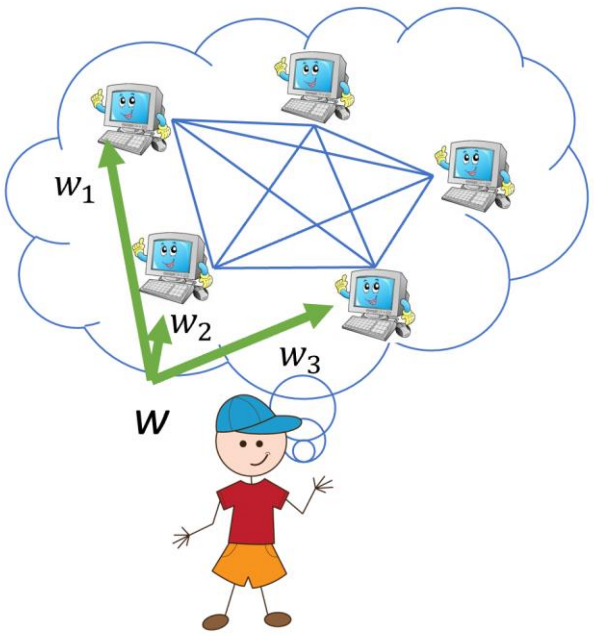
MPC-in-the-Head [IKOS08]



MPCitH uses special Client-Server-MPC



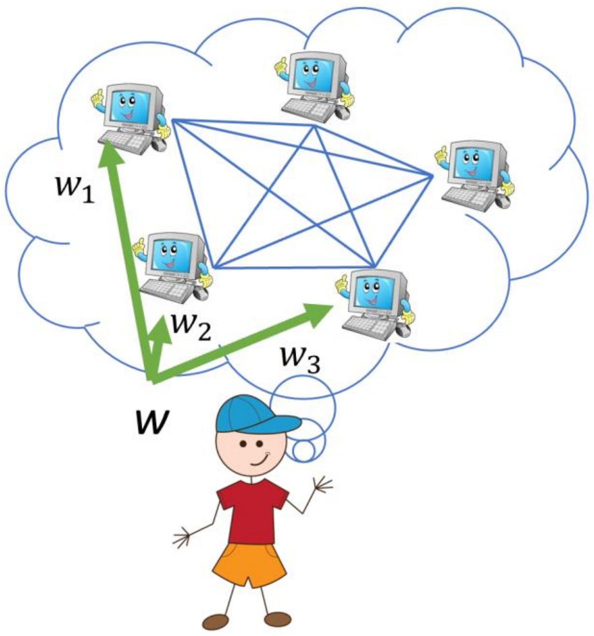
MPC-in-the-Head



Completeness

- Let C be a circuit that outputs 1 iff w is a witness for x
- Follows from Correctness of MPC

MPC-in-the-Head



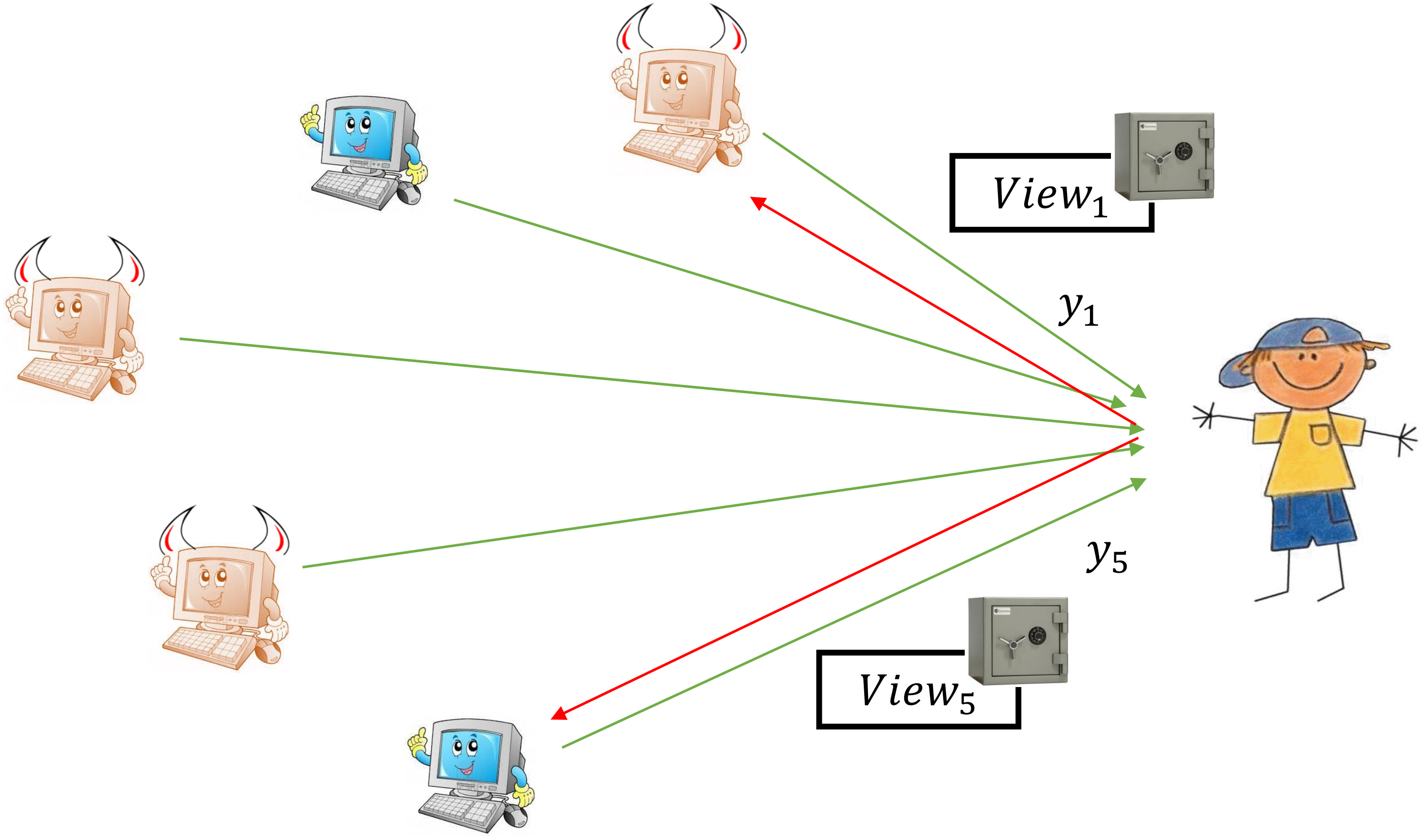
Soundness

- Prover commits to views *before* the challenge is chosen
- Must cheat in MPC protocol – some parties have to cheat (i.e. inconsistent view with honest parties)

MPC protocol is t_r-robust against cheating parties

- Prover must have cheated in $> t_r$ parties
- Combinatorial game: what's the chance the verifier doesn't open one of the $> t_r$ dishonest parties?

MPC-in-the-Head: Soundness



<u>Example</u>
MPC with $t_r = t_p = 2$

For simplicity assume only broadcast communication

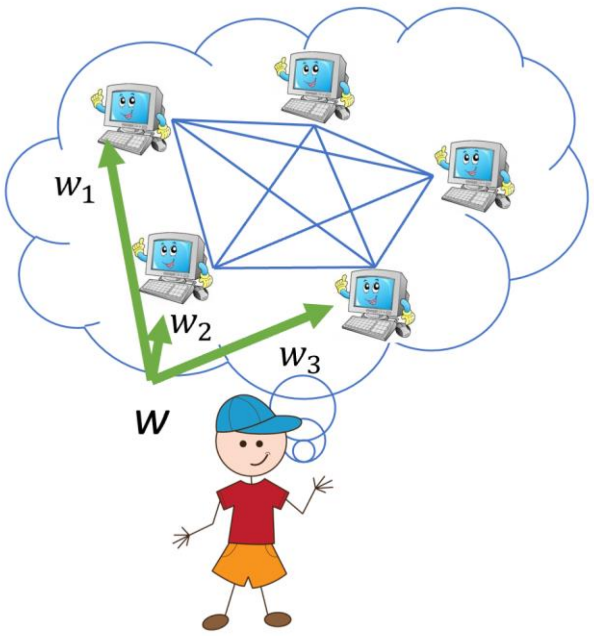
$y_1, \dots, y_5$ must reconstruct to 1

<u>All 3 dishonest parties must lie</u>

Opening one honest and dishonest party detects cheating

$$\Pr[open\ honest\ and\ dishonest | open\ two\ parties] > 1/2$$

MPC-in-the-Head



Zero-knowledge

Opening t_p views is safe due to t_p-privacy

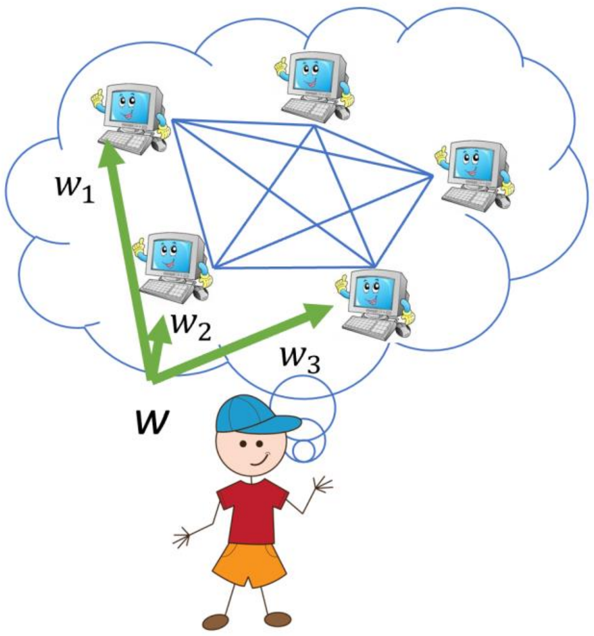
Formally

1. ZK simulato PC scheme to simulate m ad of views).

Honest Verifier-ZK: simulator knows choice of verifier in advance, can use statically secure MPC

2. Upon receiving challenge, prover *corrupts* parties in MPC simulator, obtains views and *equivocates commitments* to MPC simulator outputs

MPC-in-the-Head



Introduced in [IKOS07]

Implemented and optimized in ZKBoo [GMO16],

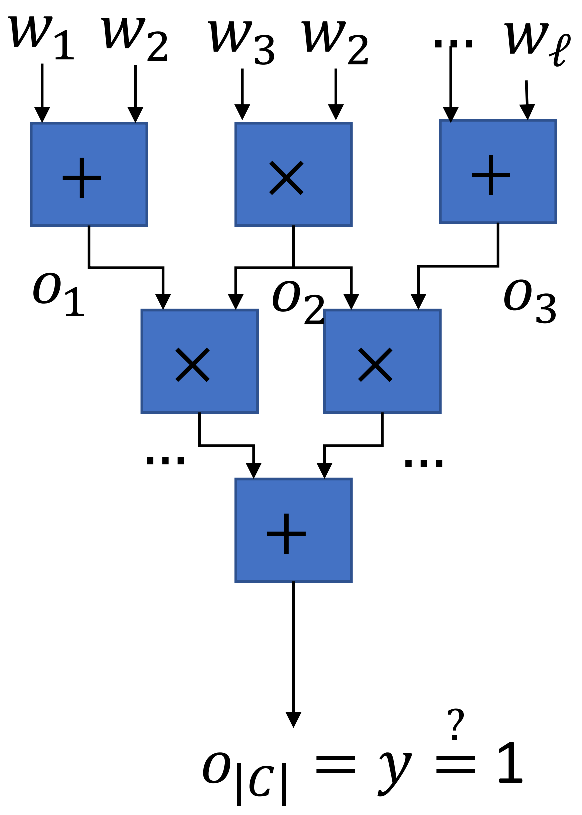
ZKB++[CDG+17]

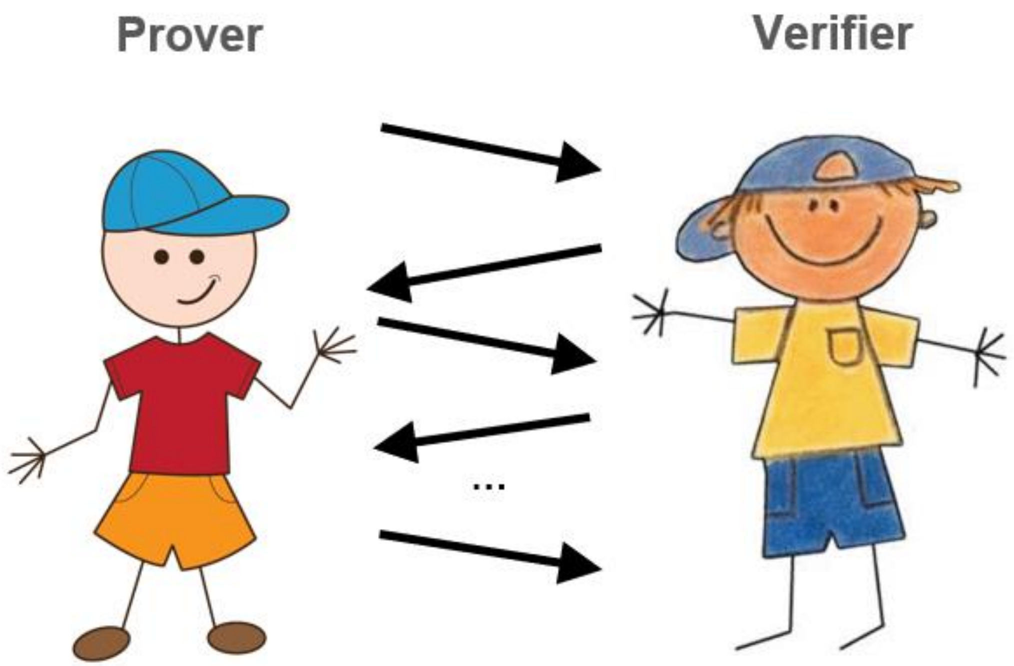
Ligero [AHIV17] – **sub-linear communication** complexity (later)!

[KKW18] – MPC-in-the-Head in the **pre-processing model**

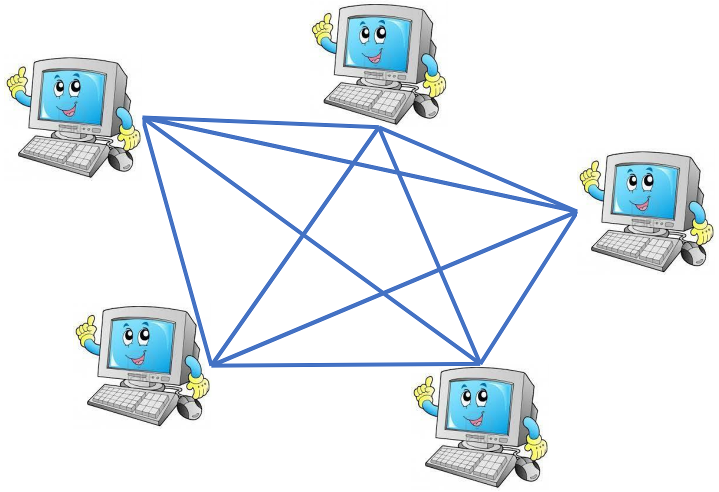
The computational model

$(x, w) \in R_L \Leftrightarrow C(w) = 1$ where $w \in \mathbb{F}^{\ell}$





MPC protocol π of [KKW18]



Circuit C over field $\mathbb{F}$

π has N parties, $t_p = N - 1$, $t_r = 0$

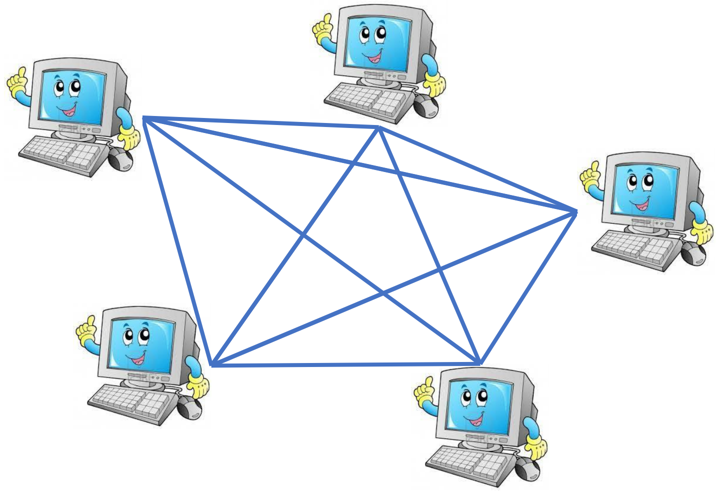
<u>Sharing of inputs $x \in \mathbb{F}$ as $[x]$:</u>

1. $P_1, \dots, P_{N-1}$ get uniformly random $x_1, \dots, x_{N-1}$
2. P_N gets $x_N = x - \sum_{i \in [N-1]} x_i$

<u>Linear operations</u>

- To compute $[\gamma] = [\alpha x + y + \beta]$ from $[x], [y]$, P_i sets share $\gamma_i \coloneqq \alpha_i x_i + y_i + \beta_i / N$

MPC protocol π of [KKW18]



Circuit C over field $\mathbb{F}$

π has N parties, $t_p = N-1$, $t_r = 0$

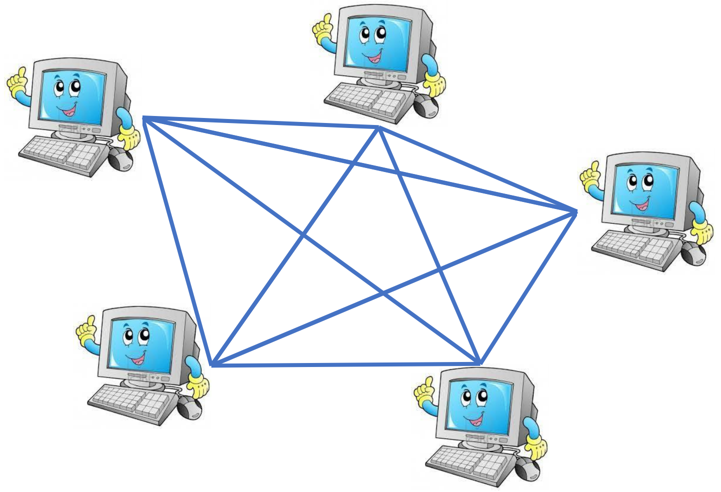
<u>Sharing of inputs $x \in \mathbb{F}$ as $[x]$:</u>

1. $P_1, \dots, P_{N-1}$ get uniformly random $x_1, \dots, x_{N-1}$
2. P_N gets $x_N = x - \sum_{i \in [N-1]} x_i$

<u>Multiplication – Beaver's trick</u>

- To multiply $[x], [y]$, assume sharing $[a], [b], [c]$ where a, b are uniformly random, $c = a \cdot b$
- Protocol:
    1. Parties reveal $[\alpha] = [x - a], [\beta] = [y - b]$
    2. Parties compute $[z] = \beta[x] + \alpha[y] - \alpha\beta + [c]$

MPC protocol π of [KKW18]



Circuit C over field $\mathbb{F}$

π has N parties, $t_p = N - 1$, $t_r = 0$

Prover always opens $N - 1$ parties, so can cheat ***only in one party***

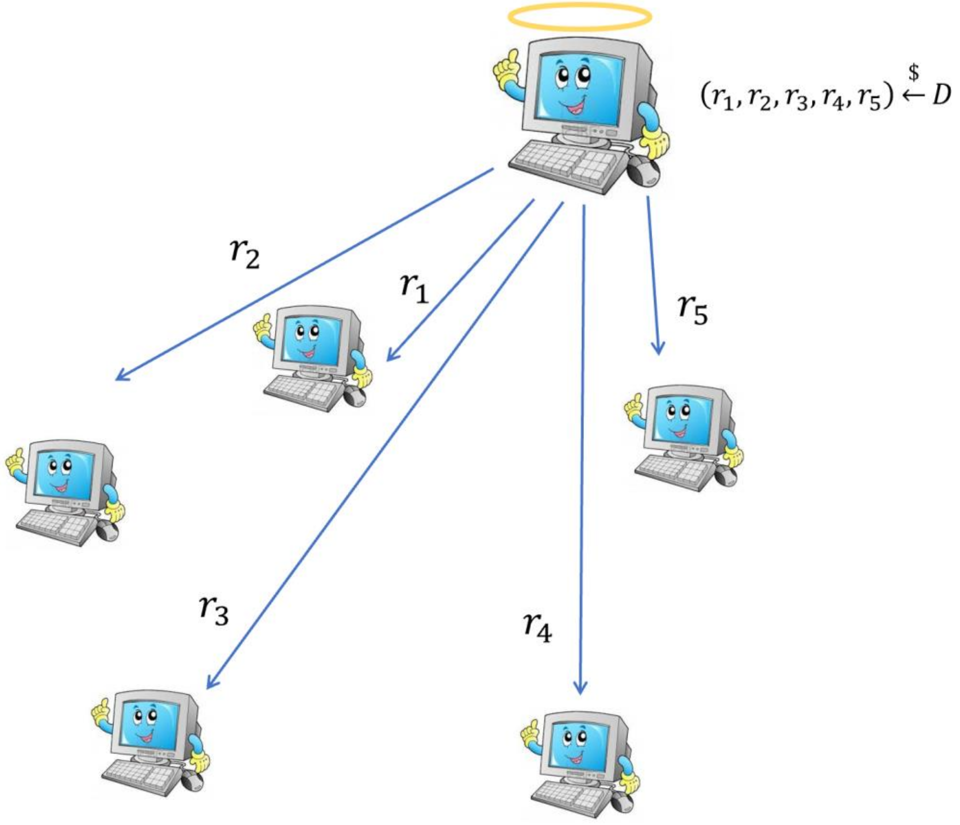
Soundness error of proof: $\frac{1}{N}$. Can decrease by *parallel repetition*.

Pre-processing in MPCitH

As part of view, each party also commits to r_i

But $r_1, \ldots, r_N$ may not be valid sharing ($c \neq a \cdot b$)

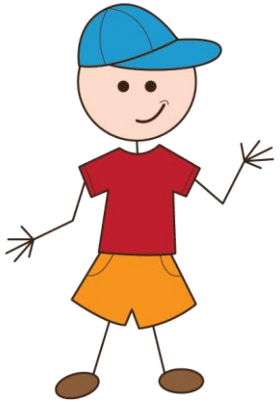
Prover has chance to cheat!



MPC-in-the-head a'la [KKW18]

Cut & Choose

Prover



Verifier

Commit to triples for MPC instances

Open subset of triples (MPC instances)

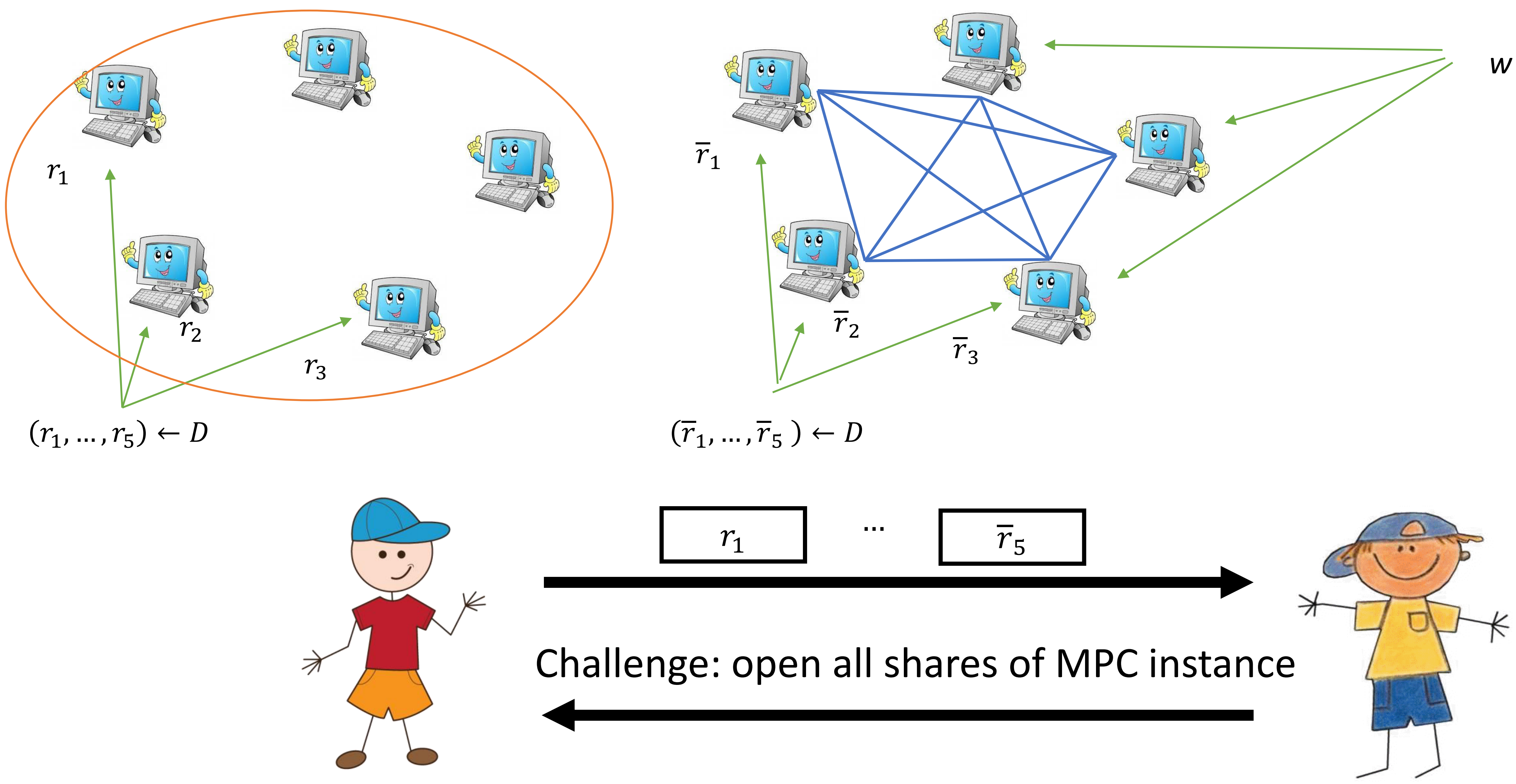
1. De-commit the chosen subset
2. Run MPC for unopened triples
3. Commit to the views of the parties

Open subset of views

De-commit the chosen views

1. Triples:
2. ...tent?
...put: correct?

Accept\Reject?

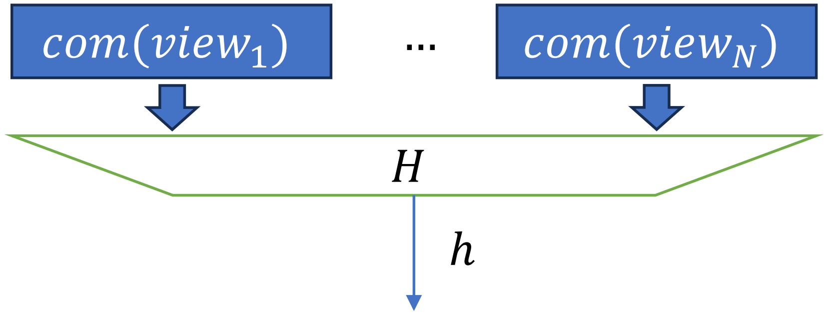

r_1
r_2
r_3
$(r_1, \dots, r_5) \leftarrow D$
w
$\bar{r}_1$
$\bar{r}_2$
$\bar{r}_3$
$(\bar{r}_1, \dots, \bar{r}_5) \leftarrow D$
r_1
…
$\bar{r}_5$
Challenge: open all shares of MPC instance

Optimizations

Vanilla protocol: Prover sends $com(view_1), \ldots, com(view_N)$

Optimization

- Prover sends $\mathrm{h} = H(com(view_1)| \cdots |com(view_N))$
- Verifier can recompute $com(view_i)$ for opened parties P_i, prover sends $com(view_j)$ for unopened parties
- Verifier checks h



Saves communication if H is CRHF

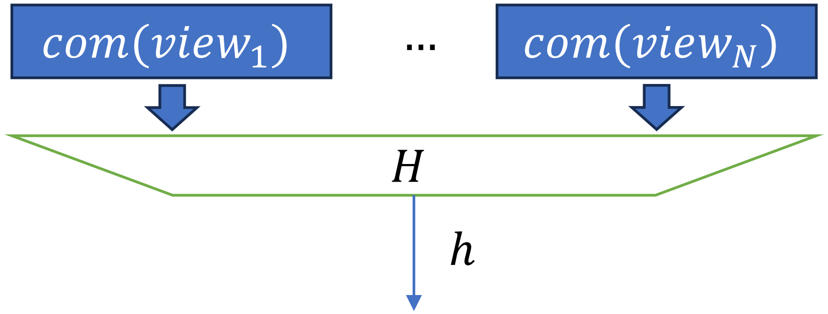
What does this save?

Vanilla protocol

N parties, τ repetitions -> $\tau \cdot N$ commitments sent

Optimization

N parties, τ repetitions -> $1 + \tau$ commitments sent



Observations about [KKW18]

Prover generates

1. Shares for inputs of parties $P_1, \dots, P_N$
2. Shares of triples for parties $P_1, \dots, P_N$

Share $[x]$:

- For $P_1, \dots, P_{N-1}$ share x_i can be uniformly random in $\mathbb{F}$
- P_N: $x_N = x - (x_1 + \cdots + x_{N-1})$

Triple $[a], [b], [c]$:

- For $P_1, \dots, P_{N-1}$ share of $[a], [b], [c]$ can be uniformly random in $\mathbb{F}$
- P_N: a_N, b_N uniformly random, $c_N = (\sum a_i) \cdot (\sum b_i) - (c_1 + \cdots + c_{N-1})$

How to generate shares randomly?

Generate shares of $P_1, \ldots, P_{N-1}$ from PRG seed $seed_i$

To open $view_i$ for $P_i \in \{P_1, \ldots, P_{N-1}\}$ prover only reveals $seed_i$ and messages obtained by P_i from other parties

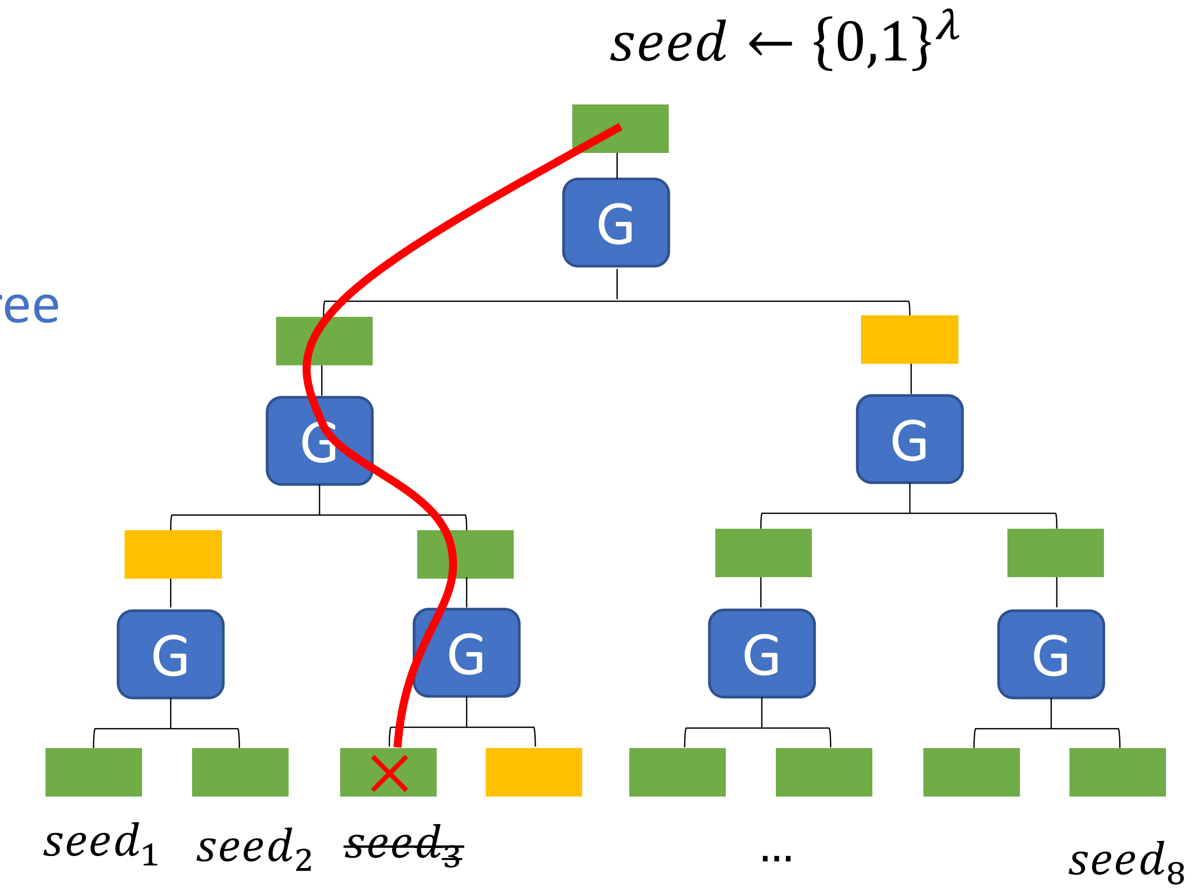
Can generate $seed_i$ from one seed $seed$: GGM trees

What is a GGM tree?

Let G be a length-doubling PRG

- Avoid sending seeds separately
  - Derive from leaves of a GGM tree

- Open $n - 1$ leaves (seeds):
  - Send $O(\log n)$ PRG seeds



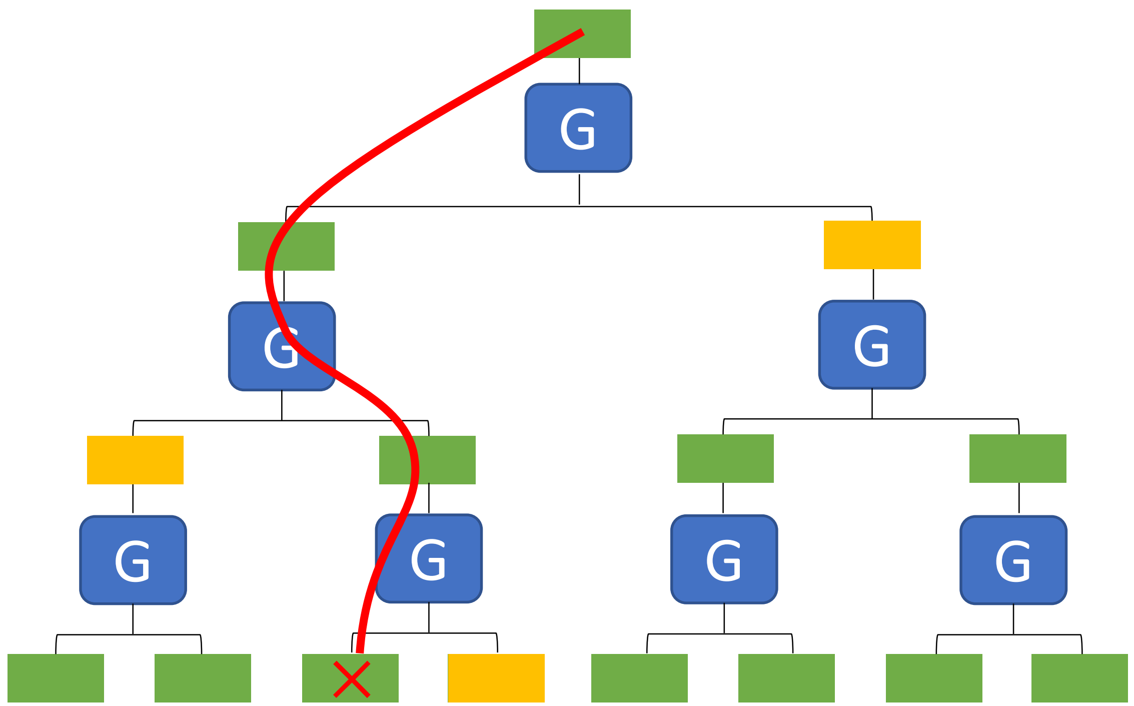
What does this save?

Vanilla protocol

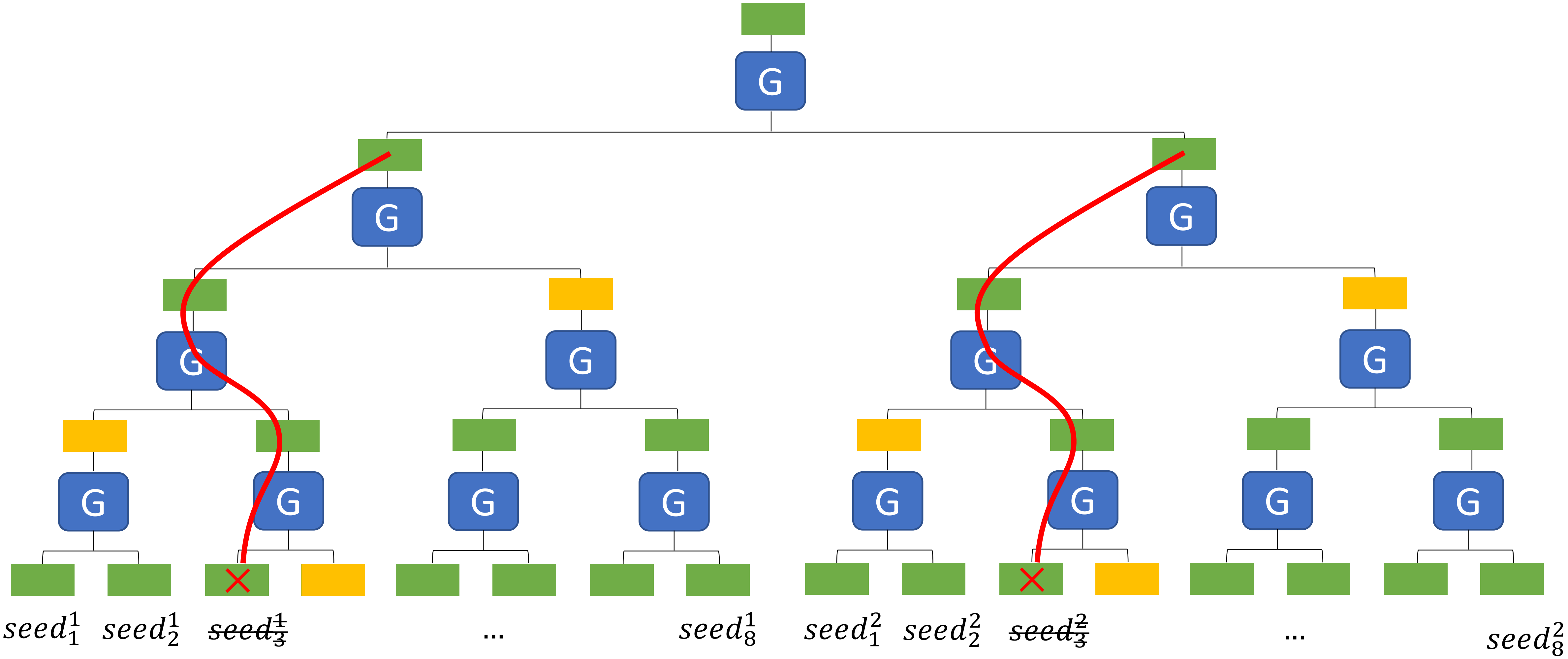
N parties, τ repetitions
-> $\tau \cdot N$ seeds

GGM optimization

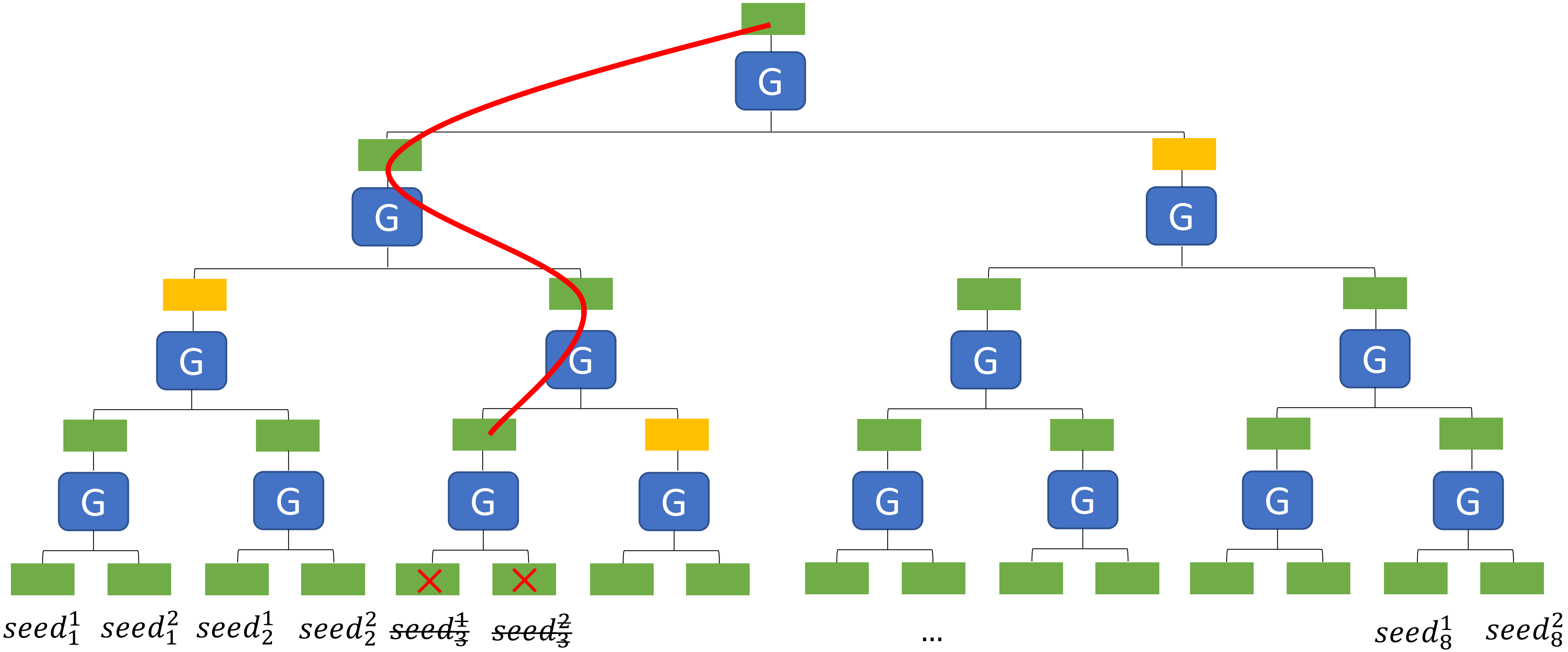
N parties, τ repetitions
-> $\tau \cdot \log(N)$ seeds



What if $\tau = 2$? Always have to open 2 paths



One-tree optimization [**B**BM+24]



What does One-tree buy you?

Proof size *depends* on challenge, can restrict to subset of challenges.

For signatures (next talk) this allows to optimize other parameters and makes prover/verifier faster.

	Sign/Verify	Size
FAEST-128s	≈ 4,4 ms	5.006 B
FAEST-128f	≈ 0,4 ms	6.336 B
FAESTER-128s	≈ 3,3 ms	4.594 B
FAESTER-128f	≈ 0,4 ms	5.444 B

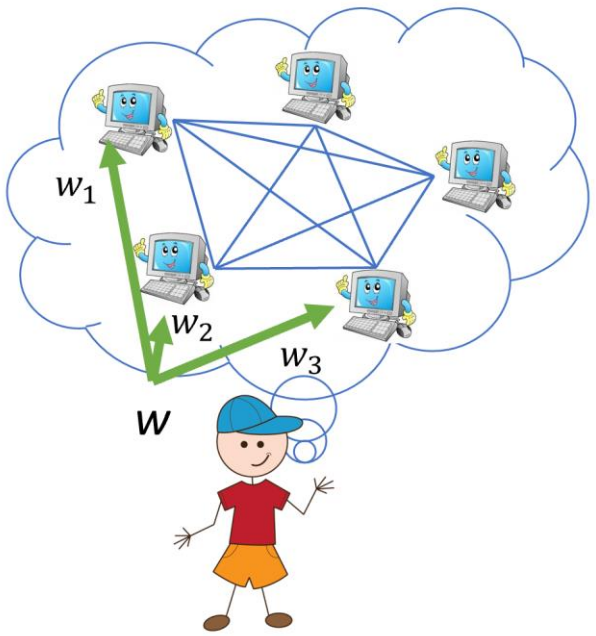
Timings on machine with AMD Ryzen 7 5800H, 3.2–4.4 GHz

Summary

What is MPC?

MPC-in-the-head: build ZK from MPC & commitments

The KKW18 construction & optimizations



Further reading

[IKOS08] Ishai, Y., Kushilevitz, E., Ostrovsky, R., & Sahai, A. (2009). Zero-knowledge proofs from secure multiparty computation.

[GMO16] Giacomelli, I., Madsen, J., & Orlandi, C. (2016). ZKBoo: Faster Zero-Knowledge for Boolean Circuits.

[CDG+17] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D. & Zaverucha, G. (2017). Post-quantum zero-knowledge and signatures from symmetric-key primitives.

[KKW18] Katz, J., Kolesnikov, V., & Wang, X. (2018). Improved non-interactive zero knowledge with applications to post-quantum signatures.

[**B**N20] Baum, C., & Nof, A. (2020). Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography.

[**B**BM+24] Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L. & Scholl, P. (2024). One tree to rule them all: Optimizing ggm trees and owfs for post-quantum signatures. *Eprint 2024/490*