Efficient MPC with an Honest Majority

Yuval Ishai
Technion

Advertisement: TPMPC 2020

=== Theory & Practice of Multi-Party Computation Workshop 2020 ===

The TPMPC workshops aim to bring together practitioners and theorists working in multi-party computation. This year's event will be held in Aarhus, Denmark from May 25th to May 28th.

Call for Contributed Talks

Deadline: 25 February 2020

TPMPC solicits contributed talks in the area of the theory and/or practice of secure multiparty computation. Talks can include papers published recently in top conferences, or work yet to be published. Areas of interest include:

- Theoretical foundations of multiparty computation: feasibility, assumptions, asymptotic efficiency, etc.
- Efficient MPC protocols for general or specific tasks of interest
- Implementations and applications of MPC

For further details regarding contributed talks and submissions, see:
https://www.multipartycomputation.com/tpmpc-2020

MPC with an Honest Majority

- **Several potential advantages**
  - Unconditional security
  - Guaranteed output and fairness
  - Universally composable security with no setup
  - This talk: efficiency
- **Main feasibility results**
  - Perfect security with $t<n/3$ [BGW88,CCD88]
  - Statistical security with $t<n/2$ (over broadcast) [RB89]
- **Goal: IT security with minimal complexity**
  - Communication
  - Computation
  - Rounds

Where is IT MPC stuck?

Ideal goal: security for free

In reality…

Even for passive security, even when t << n

- Communication: can't beat circuit size
  - Except for "very structured" or "very complex" functions
  - 3-party case: $\sim 2^{\sqrt{|x|}}$ via 3-server PIR [Efr09,BIKK14]
- Computation: can't get constant overhead
  - Except when t=O(1)
- Rounds: can't significantly beat circuit depth
  - Except for functions that are "not too complex"
  - Benny's talk…

Can we do better with (comp.) 2PC?

Even worse.

Passive: Boolean+arithmetic
[IKOS08, ADINZ18]
Using poly-stretch local PRGs

Active: only arithmetic
[BCGGHJ17, BCGI18]

Yes we can, using FHE [Gen09] or HSS [BGI16], but with big concrete overhead

t circuit size

or "very complex" functions

- 3-party case: ~ via 3-server PIR [Efr09,BIKK14]

- **Computation: c n't get constant overhead**
  - Except when t=O(1)
- **Rounds: can't significantly beat circuit depth**
  - Except for functi that are "not too complex"
  - Benny's talk…

Yes we can, using garbled circuits [Yao86], even with low communication via FHE or HSS

What can we realistically hope for?

Active security Big # parties n	≈ Total cost	**Best known** Passive security O(1) parties t=1

- **Optimal resilience**
- **Communication**: O(|C|)
- **Computation**: polylog(n) overhead
- **Rounds**: O(depth)

What can we get?

Active security
Big # parties n

$\approx$
Total cost

Best known
Passive security
O(1) parties
t=1

- Near-optimal resilience
  - E.g., $t<0.33n$ perfect, $t<0.49n$ statistical
- Communication: $O(|C|)$
  - Assuming $n \ll |C|$, $\text{depth}(C) \ll |C|$
- Computation: **polylog(n) overhead** (log for arithmetic)
- Rounds: **O(depth)**

What can we get?

Active security $\approx$ **Best known** Passive security

- Needed for eliminating O(n) overhead [DLN19]
- Good price to pay when n is big
  (e.g., repetition vs. asymptotically good code)
- Makes almost no difference for "MPC in the head"

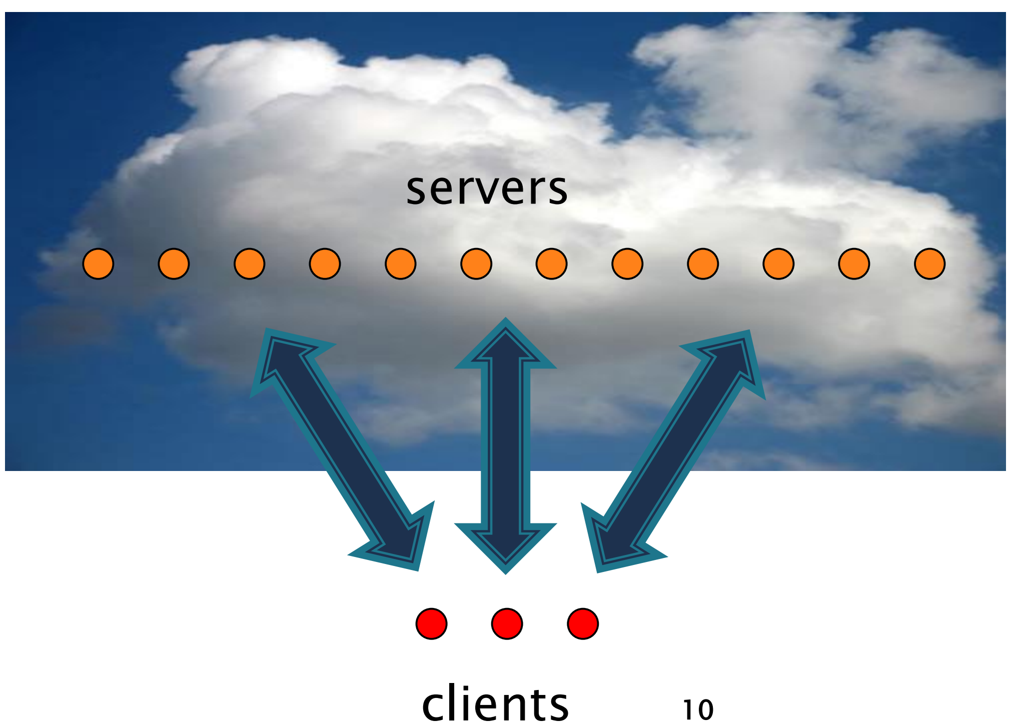
- **Near-optimal resilience**
  - E.g., $t<0.33n$ perfect, $t<0.49n$ statistical
- **Communication**: **O(|C|)**
  - Assuming $n \ll |C|$, $\text{depth}(C) \ll |C|$
- **Computation**: **polylog(n) overhead** (log for arithmetic)
- **Rounds**: **O(depth)**

What can we get?

- **This talk: several simplifying assumptions**
  - Inputs originate from a constant number of "clients"
  - Security with abort
  - Statistical security against static active adversary
  - Small fractional resilience
  - Broadcast
- **Assumptions can be eliminated**

The model

- **m≥2 clients, n servers**
  - Only clients have inputs and outputs
  - Assume m=O(1) in most of this talk
  - Motivated by “MPC in the head” (next talk)
  - Results extend to standard n-party model



The model

- Synchronous secure point-to-point channels + broadcast
  - Servers only talk to clients
- Active, static adversary corrupting:
  - at most cn servers for some constant $0<c<1/2$
  - any subset of the m clients
- Statistical security with abort

Some literature pointers

- Hirt-Maurer 01, Damgård-Nielsen 07, Beerliova-Hirt 08, BenSasson-Fehr-Ostrovsky 12, Genkin-I-Prabhakaran-Sahai-Tromer 14, I-Kushilevitz-Prabhakaran-Sahai-Yu 16, Cascudo-Cramer-Xing-Yuan 18, Chida-Genkin-Hamada-Ikarashi-Kikuchi-Lindell-Nof 18, …
  - n-party perfect/statistical MPC with optimal resilience
  - Total communication scales (almost) linearly with n

- Damgård-I 06, I-Prabhakaran-Sahai 09
  - O(1)-client n-server statistical MPC with near-optimal resilience
  - Total communication insensitive to n
  - Total computation scales with log(n)
    (x statistical-security parameter in Boolean case)

- Damgård-I-Kroigaard-Nielsen-Smith 08, Damgård-I-Kroigaard 10
  - Essentially the same for perfect MPC in standard n-party model

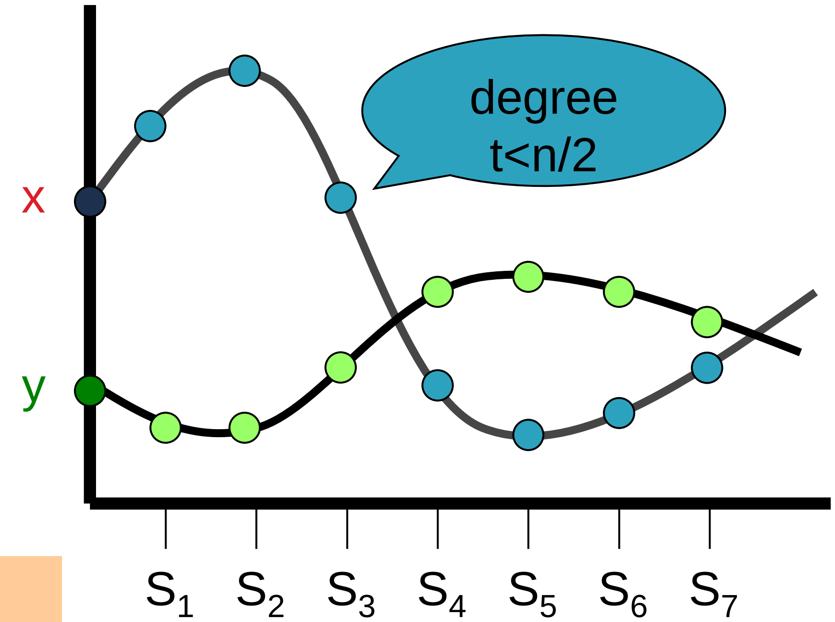
Some literature pointers

- Bracha 87
  - Using committees to boost security threshold
- Franklin-Yung 92
  - Share packing technique
- Chen-Cramer 06
  - Using constant-size fields via AG codes
  - Helps reduce communication for Boolean circuits
- … Araki-Furukawa-Lindell-Nof-Ohara16 …
  - Different line of work
  - Minimizing concrete overhead for a small number of parties

  … more in Niv's talk

Starting point: BGW/CCD

- **Secret-share inputs**
- **Evaluate C on shares**
  - Non-interactive addition
  - Interactive multiplication
- **Recover outputs**

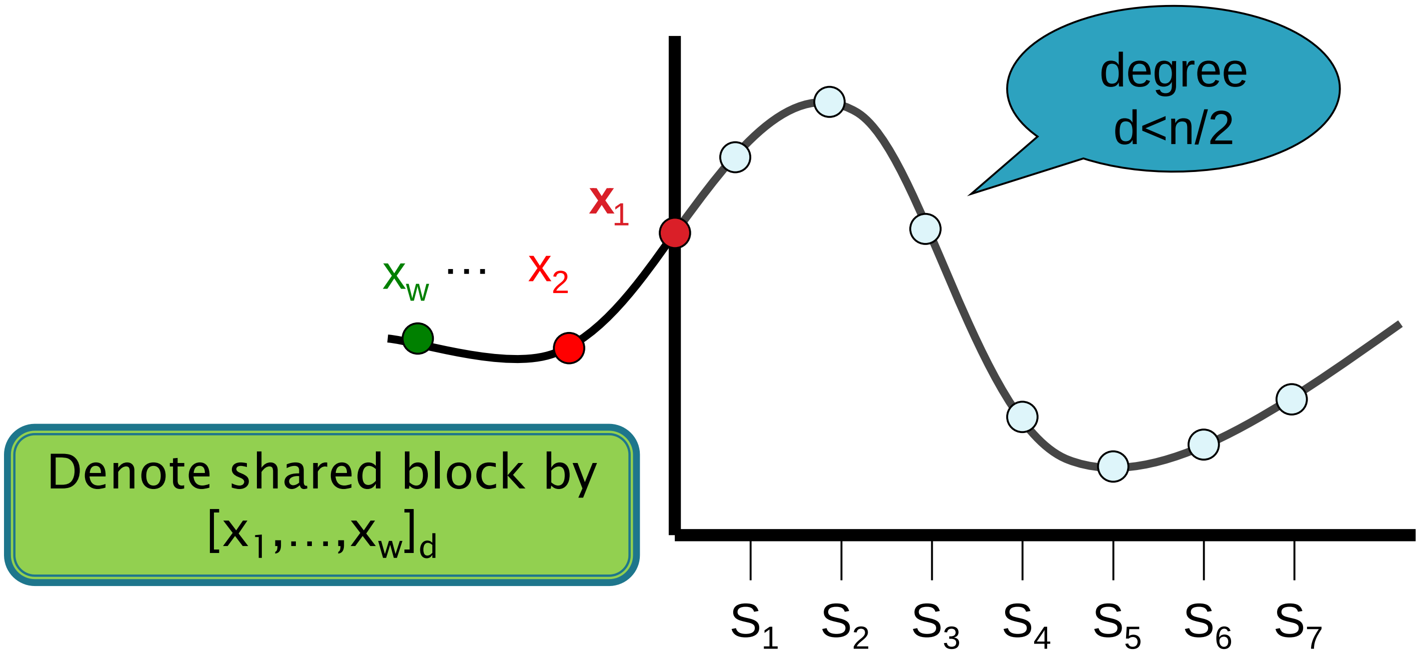


- Secure with $t<n/2$ (passive
  or $t<n/3$ (active)
- Complexity: $|C|\cdot O(n^2)$ (passive)
  $|C|\cdot poly(n)$ (active)

Sources of overhead

- **Each wire value is split into n shares**
  - Use "packed secret sharing" to amortize cost
- **Multiplication involves communication between each pair of servers**
  - Reveal blinded products to a single client
- **Expensive consistency checks**
  - Efficient batch verification

Share packing



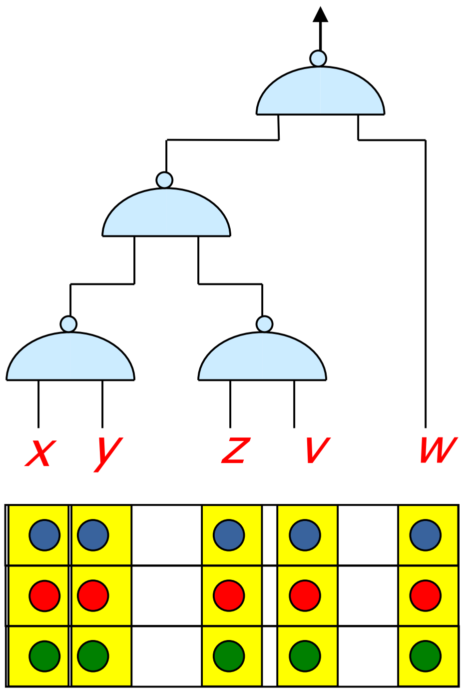
- Handle block of w secrets for price of one.
- Security threshold degrades from d to d-w+1
- w=n/10 ➔ Ω(n) savings for small security loss
- Compare with error correcting codes

BGW with share packing?

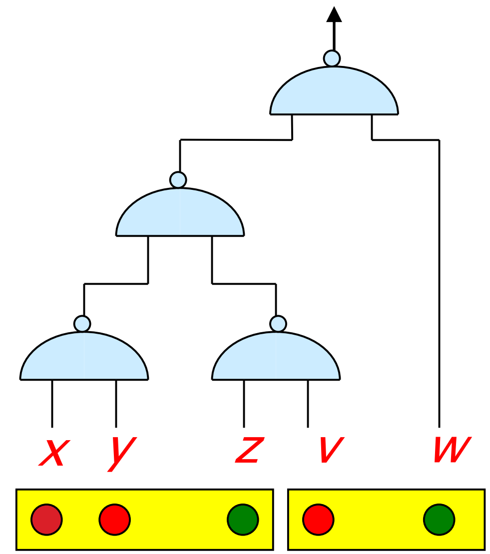
YES: evaluate a circuit on multiple inputs in parallel

NO: evaluate a circuit on a single input

3 inputs

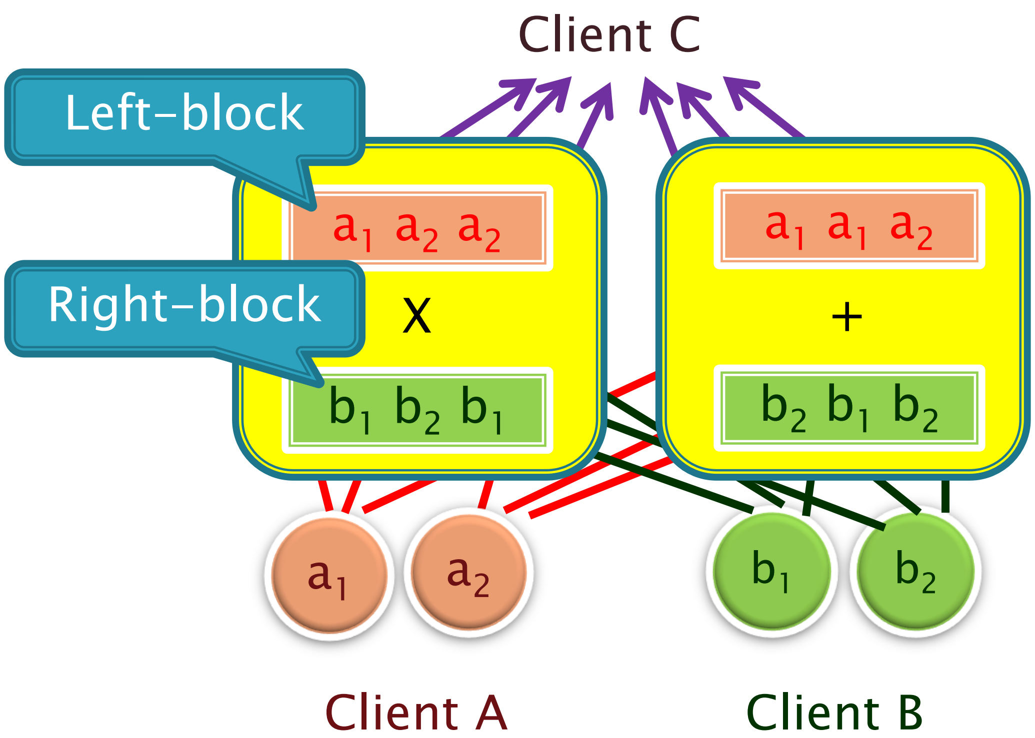


5 blocks



"SIMD-friendly" computation

Warmup: Passive, depth 1



A→S: $p_A=[a_1,a_2,a_2]_d$
$q_A=[a_1,a_1,a_2]_d$
$z_A=[0,0,0]_{2d}$

B→S: $p_B=[b_1,b_2,b_1]_d$
$q_B=[b_2,b_1,b_2]_d$
$z_B=[0,0,0]_{2d}$

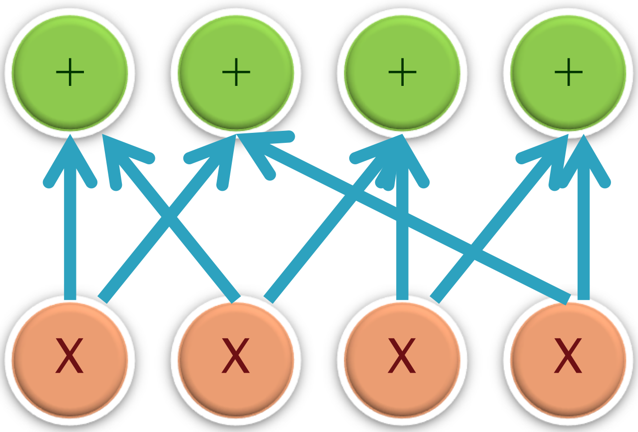
S→C: $p_A p_B+z_A+z_B$
q_A+q_B

- Extends to constant-depth circuits
- Still 2 rounds, $t=\Omega(n)$

Passive, any depth

- **Assume circuit is composed of layers 1,…,H.**
- **Clients share inputs into $[left^1]_d$ and $[right^1]_d$**
- **For h=1 to H-1:**
  - Clients generate random blocks $[r]_{2d}$, $[left\_r]_d$ and $[right\_r]_d$ replicated according to structure of layer h+1
  - Servers send masked output shares of layer h to Client A: $[y]_{2d}=[left^h]_d*[right^h]_d+[r]_{2d}$ ($* \in \{x,+,-\}$)
  - A decodes, rearranges and reshares y into $[left\_y]_d$, $[right\_y]_d$
  - Servers let
    - $[left^{h+1}]_d=[left\_y]_d-[left\_r]_d$
    - $[right^{h+1}]_d=[right\_y]_d-[right\_r]_d$
- **Servers reveal output shares $[left^H]_d*[right^H]_d+[0]_{2d}$**

Example



$y_1\ y_1\ y_2\ y_3$ – $r_1\ r_1\ r_2\ r_3$ → $Left^{h+1}$

$y_2\ y_4\ y_3\ y_4$ – $r_2\ r_4\ r_3\ r_4$ → $Right^{h+1}$

$Left^h$ X $Right^h$

+ $r_1\ r_2\ r_3\ r_4$

$y_1\ y_2\ y_3\ y_4$

Active security

- **Need to protect against $t=\Omega(n)$ malicious servers and $t' < m$ malicious clients.**
- **Malicious servers handled via error correction**
  - Valid shares form a good error-correcting code
  - Error detection sufficient for security with abort
- **Malicious clients handled via efficient VSS procedures (coming up)**

Efficient statistical VSS

- **Recall: only shoot for security with abort**
- **Two types of verification procedures**
  - Verify that shares lie in a linear space
    - E.g., degree-d polynomials
  - Verify that shared blocks satisfy a given replication pattern
    - E.g., $[r_1,r_1,r_2,r_1]$ $[r_2,r_3,r_1,r_2]$
- **Cost is amortized over multiple instances**

Verifying membership in a linear space

- **Suppose Client A distributed a vector v between servers.**
  - S_i holds the i-th entry of v
  - Can be generalized to an arbitrary partition of entries
- **Goal: Prove in zero-knowledge to Client B that v is in some (publicly known) linear space L over F.**
- **Protocol:**
  - A distributes a random $u \in_r L$
  - B picks and broadcasts $c \in_r F$
  - Servers jointly send $w = cv + u$ to B
  - B checks that $w \in L$
- **ZK: w is a random vector in L**
- **Soundness** (static corruption):
  - consider messages from honest servers
  - $cv+u,\ c'v+u \in L \Rightarrow (c-c')v \in L \Rightarrow v \in L$
  - soundness error $\leq 1/|F|$

Amortizing cost

- Can be jointly generated by clients
- Can be pseudorandom
  Unconditional PRG suffices

$c_1 \times v_1$

$c_2 \times v_2$

$c_3 \times v_3$

$c_4 \times v_4$

$c_5 \times v_5$

$+\ u$

$w \in L$?

Adaptive security:

- Needed for ZK/2PC application
- Union bound too loose
- Tighter analysis: AHIV17,…

Verifying replication pattern

secret	a b c d	e f g h
	inner product	
public	pub1	pub2

1. Write replication requirement as linear equations:

b−a=0
e−b=0
g−e=0
h−g=0
d−c=0
f−d=0

2. Take random linear combination:

r1*(b−a)+
r2*(e−b)+
r3*(g−e)+
r4*(h−g)+
r5*(d−c)+
r6*(f−d) = 0

3. Bring to a <secret , public > format

Verifying replication pattern

secret | a b c d | e f g h

inner product

public | pub1 | pub2

a b c d × pub1 + e f g h × pub2

+ $z_1\ z_2\ z_3\ z_4$

Random block with sum 0
Generated by prover

Asymptotic efficiency

- **Communication**
  - O(|C|) field elements (|F|>n) + “low order terms”
  - Low order terms include:
    - Additive term of O(depth·n) for layered circuits
    - depth ➔ # “communicating layer pairs” for general circuits
    - Multiply by k/log|F| for small fields
      (k = statistical security parameter)
- **Computation**
  - Communication x O(log n)
    - Uses FFT for polynomial operations
  - Multiply by k/log|F| for small fields

Boosting security threshold

- **Goal: small fractional resilience ➔ nearly optimal resilience**
  - without increasing asymptotic complexity!
- **Solution: Bracha-style server virtualization**
  - Example: 0.01n-secure Π ➔ 0.33n-secure Π'
  - Pick n committees of servers such that
    - Each committee is of size s=O(1)
    - If 0.33n servers are corrupted, then > 99% of the committees have < s/3 corrupted members
    - Choose committees at random, or use explicit constructions
- **Π' uses s-party BGW to simulate each server in Π by a committee**
  - Overhead poly(s)=O(1)

Using constant-size fields

- Consider a boolean circuit C with |C|» depth
- Previous protocol requires |F|>n
  - O(|C| logn) bits of communication
- Can we get rid of the logn term?
- Yes, using algebraic-geometric codes
  - Field size independent of n
  - Small fractional loss of resilience

Other extensions

- **Many clients**
  - Previous protocol required generating secret blocks
  - Easy to implement by summing blocks generated by all clients
  - Overhead can be amortized if only a constant fraction of clients are corrupted
    - Use routing network to convert circuit into regular form
    - Replace summing blocks by better randomness extraction
  - Gives protocols with polylog(n) overhead in standard n-party setting with $t=\Omega(n)$.

- **Perfect security**
  - Use efficient variant of BGW VSS with share packing
  - Alternatively: "hyperinvertible matrix" approach [BH08]

Conclusions

- Honest-majority MPC protocols are efficient!
  - Total communication = O(|C|) (+ low-order terms)
    - At most polylog(|C|) overhead with n clients
  - Total computation O~(|C|)

  - Relevant to MPC with dishonest majority (next talk)

- Open efficiency questions
  - Break circuit size communication barrier for IT security
  - Constant computational overhead for $t=\Omega(n)$