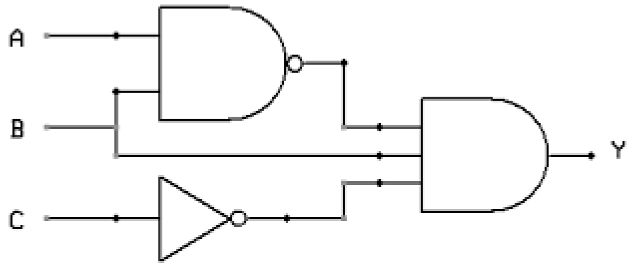
LevioSA: Lightweight Secure Arithmetic Computation from Any Passively Secure OLE

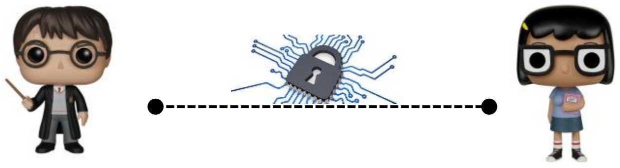
Carmit Hazay (Bar-Ilan University)
Yuval Ishai (Technion)
Antonio Marcedone (Cornell Tech)
Muthuramakrishnan Venkitasubramaniam
(U. of Rochester / Cornell Tech)

Secure Computation

How is a function represented?

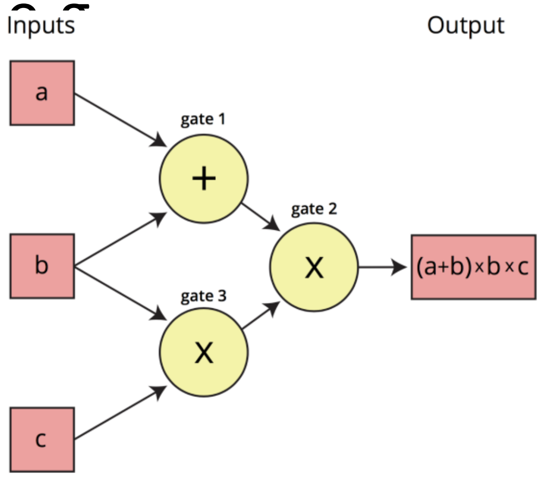
Classically, Boolean circuits [Yao86, GMW87,…]





Arithmetic Computation

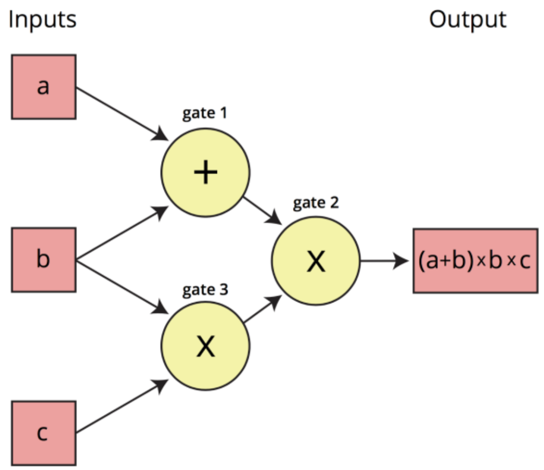
- Many computations are done over an arbitrary field $\mathbb{F}$
  - Mixing arithmetic with Boolean, e.g. machine learning
  - Arithmetic computation with "non-arithmetic" inputs, bit decomposition [LPSY15]

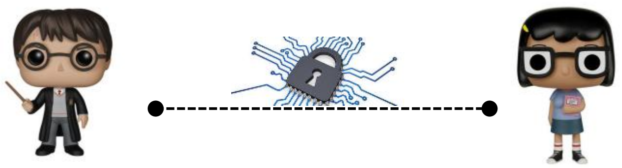


- Notable examples:
  - SHA-256
  - Threshold cryptography [BF97, Gil99…]
  - Machine learning [LP00,…, JVC18, MR18, WCG18]
  - Pattern matching [HL08, HT10, …,KRT17]
  - Even BMR garbling [LPSY15,…]

This Talk

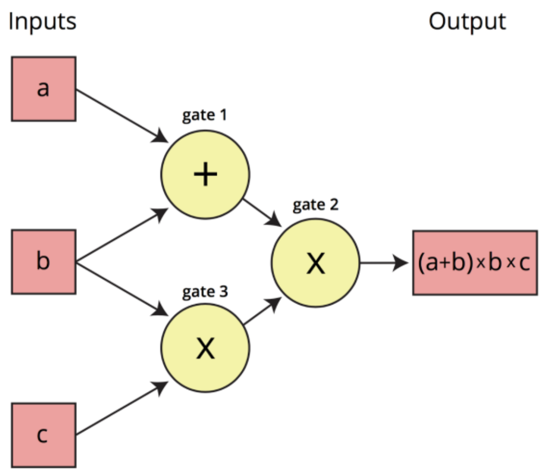
- Two-party
- Active security
- Arithmetic circuits for any field



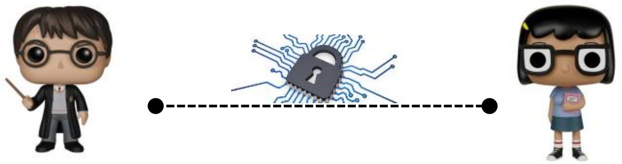


This Talk

- Two-party
- Active security
- Arithmetic circuits for any field



Motivating question:
Overhead for **active security**
given black-box access to
any passive secure OLE implem.



What is OLE?

Oblivious linear evaluation (OLE)

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT

6 $\log(|\mathbb{F}|)$ calls to active OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT
   - Requires bit-decomposition

$6 \log(|\mathbb{F}|)$ calls to active OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT
   - Requires bit-decomposition

6 $\log(|\mathbb{F}|)$ calls to active OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]

9 kbit per auth. triple

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT
   - Requires bit-decomposition

6 $\log(|\mathbb{F}|)$ calls to active OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]
   - Optimizing lattice based construction

9 kbit per auth. triple

Current Approaches to Practical Arithmetic 2PC

1. 2PC in the OLE-hybrid [GMW87, IPS09, DGNNR17]
   - Black-box calls to OLE

22 calls to active OLE

2. 2PC in the OT-hybrid [Gil99, KOS16, FPY18]
   - Black-box calls to OT
   - Requires bit-decomposition

$6 \log(|\mathbb{F}|)$ calls to active OT

3. 2PC based on semi-homomorphic encryption [BDOZ11, DPSZ12, KPR18]
   - Optimizing lattice based construction

9 kbit per auth. triple

Main Result

Theorem 1: Actively secure 2PC for most functions that makes **O(1)** black-box calls to **passive OLE** protocol per multiplication

First efficient implem. of **general** passive-to-active compiler [ala IPS08]

Main Result

Theorem 1: Actively secure 2PC for most functions that makes **O(1)** black-box calls to **passive OLE** protocol per multiplication

First efficient implem. of **general** passive-to-active compiler [ala IPS08]

Best passive: GMW 2 calls to passive OLE protocol,

For "nice" circuits our communication overhead is 2

Main Result

Theorem 1: Actively secure 2PC for most functions that makes **O(1)** black-box calls to **passive OLE** protocol per multiplication

First efficient implem. of **general** passive-to-active compiler [ala IPS08]

Best passive: GMW 2 calls to passive OLE protocol,

For "nice" circuits our communication overhead is 2

[DGNNR17] makes **22** black-box calls to any **active OLE** for any function

and **44** calls to specific RS-based passive OLE [GNN17]

Main Result

Theorem 1: Actively secure 2PC for most functions that makes **O(1)** black-box calls to **passive OLE** protocol per multiplication

First efficient implem. of **general** passive-to-active compiler [ala IPS08]

Best passive: GMW 2 calls to passive OLE protocol,

For "nice" circuits our communication overhead is 2

[DGNNR17] makes **22** black-box calls to any **active OLE** for any function and **44** calls to specific RS-based passive OLE [GNN17]

Corollary [Thm 1]: **16** black-box calls to any **passive OLE** for auth. triples

Main Result

Theorem 2: Active OLE that makes **2** black-box calls to **any passive OLE** protocol in the batch setting

[GNN17] constructs active OLE via 2 calls to a specific passive OLE

Noisy RS assumption forces communication overhead at least 32 field elements

Black-Box Use of *Any* **Passive OLE**

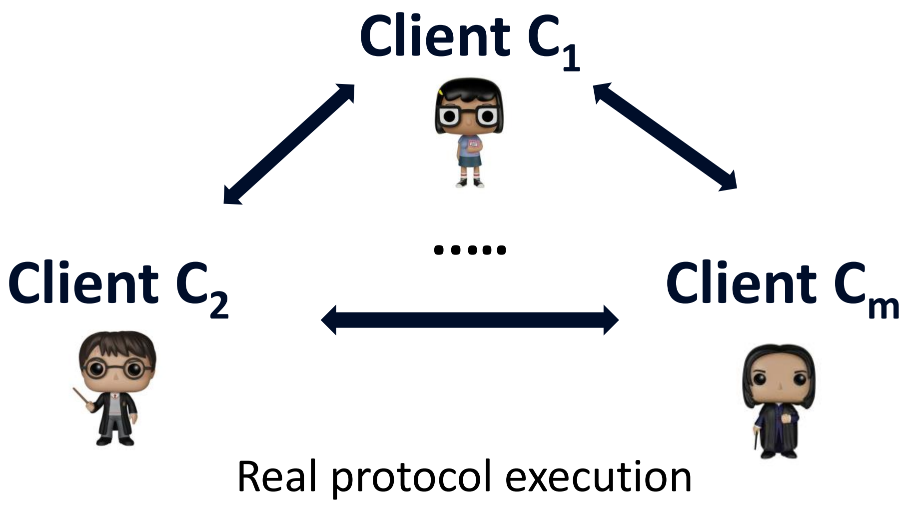
1. **More flexibility**
   - Use any existing approach to passive OLE (e.g., lattice-based, group-based, code-based, etc.)
     - Does not need "ZK friendliness"
   - Off-the-shelf software/hardware implementation

2. **Bonus feature:** "error-correct" weak implem. of passive OLE efficiently [in progress]
   - Constant correctness error (group-based HSS schemes [BGI16])
   - Constant privacy error (aggressive params. for lattice-based OLE)

Underlying Technique: MPC-in-the-Head [IKOS07, IPS08]

Two building blocks:

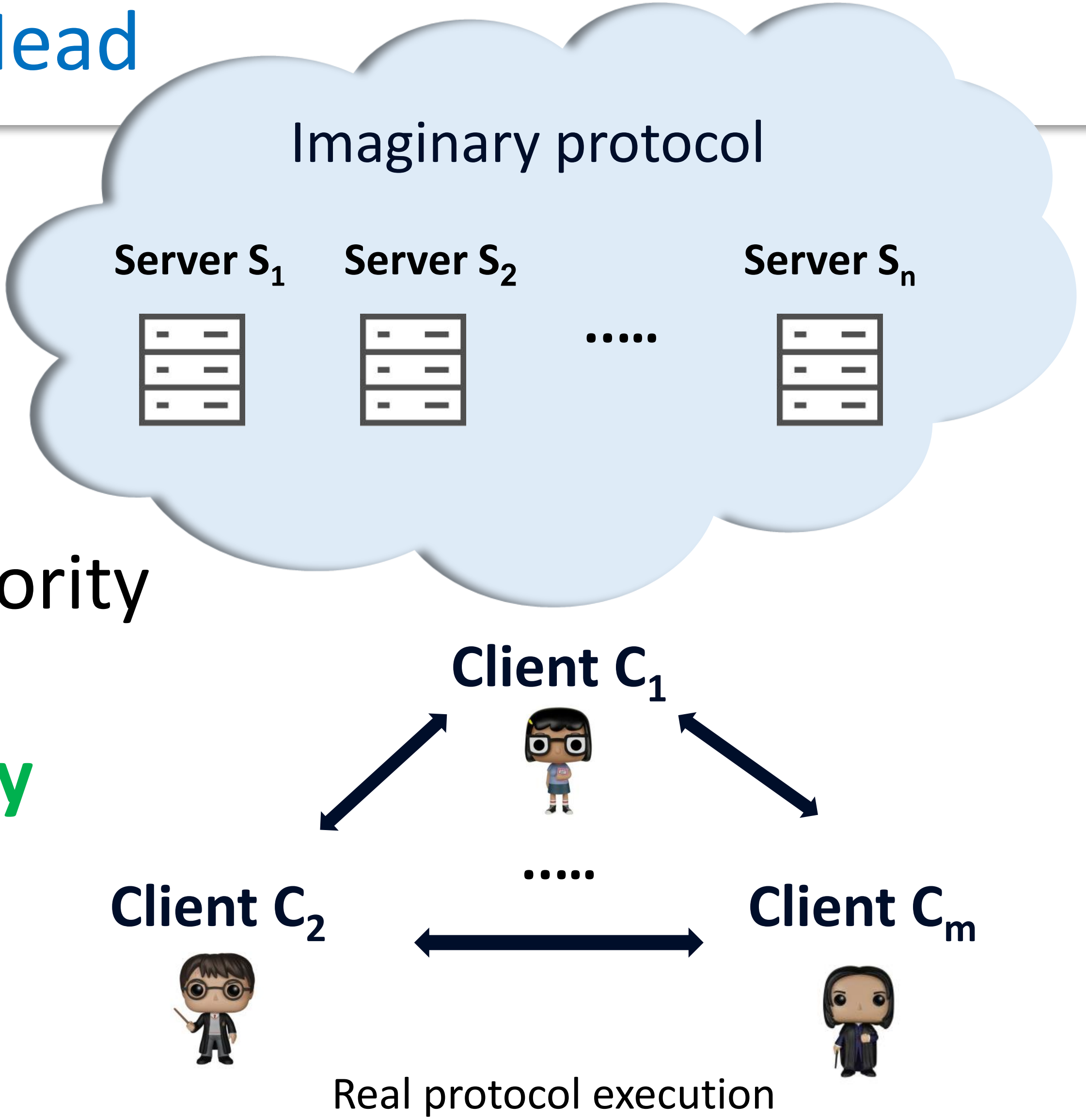
1. **Passive MPC** with dishonest majority
   - Namely, inner protocol
2. **Active MPC** with **honest majority**
   - Namely, outer protocol



Underlying Technique: MPC-in-the-Head

Two building blocks:

1. **Passive MPC** with dishonest majority
   - Namely, inner protocol
2. **Active MPC** with **honest majority**
   - Namely, outer protocol

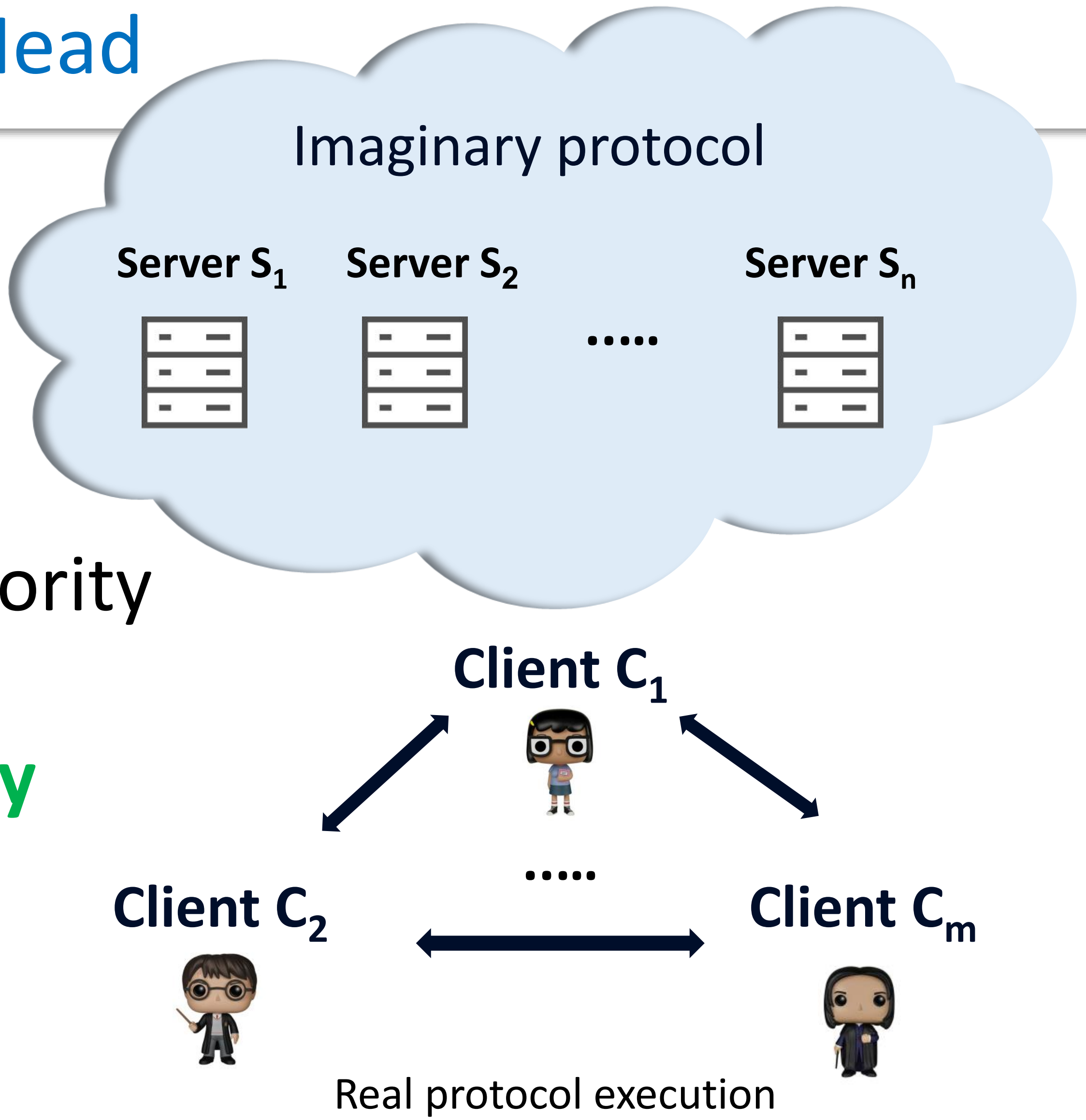


Underlying Technique: MPC-in-the-Head

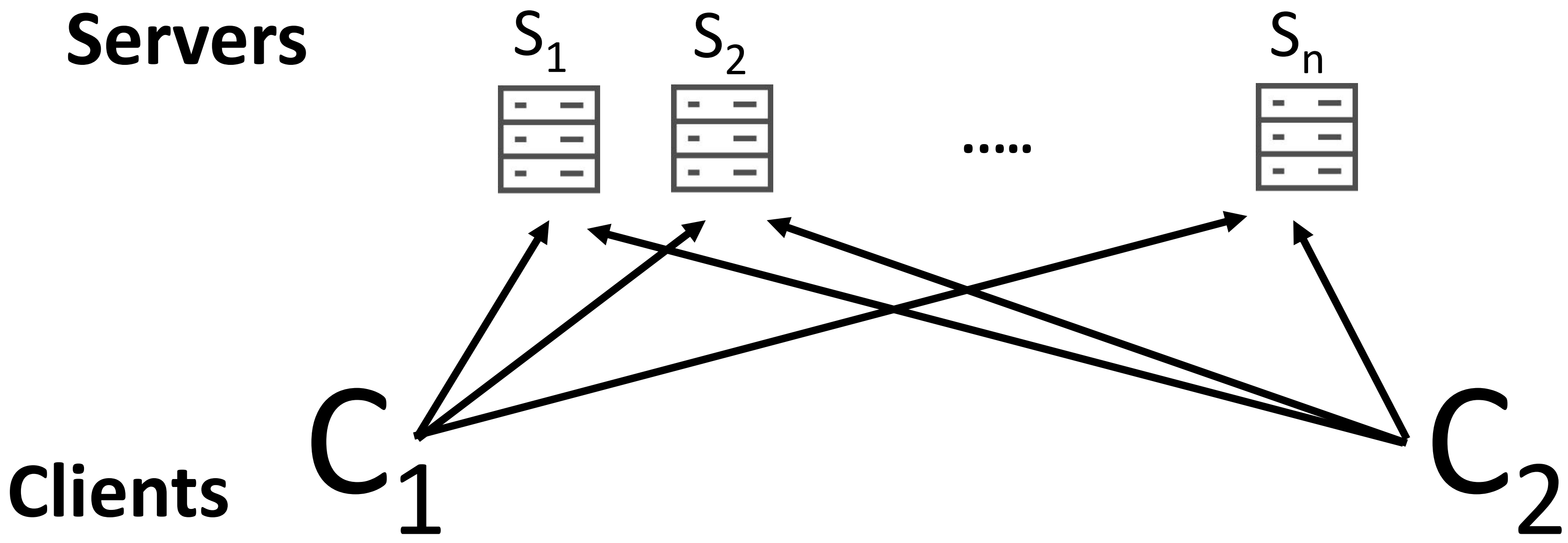
Two building blocks:

1. **Passive MPC** with dishonest majority
   - Namely, inner protocol
2. **Active MPC** with **honest majority**
   - Namely, outer protocol

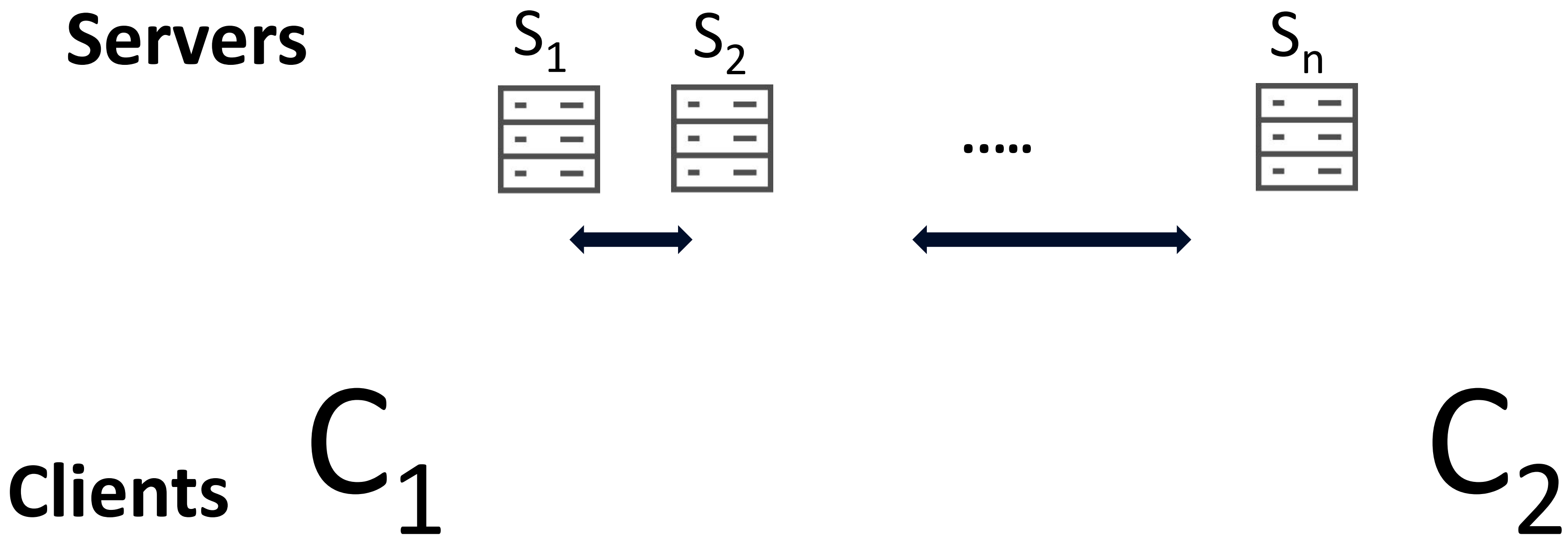
Utilizing best of both worlds!



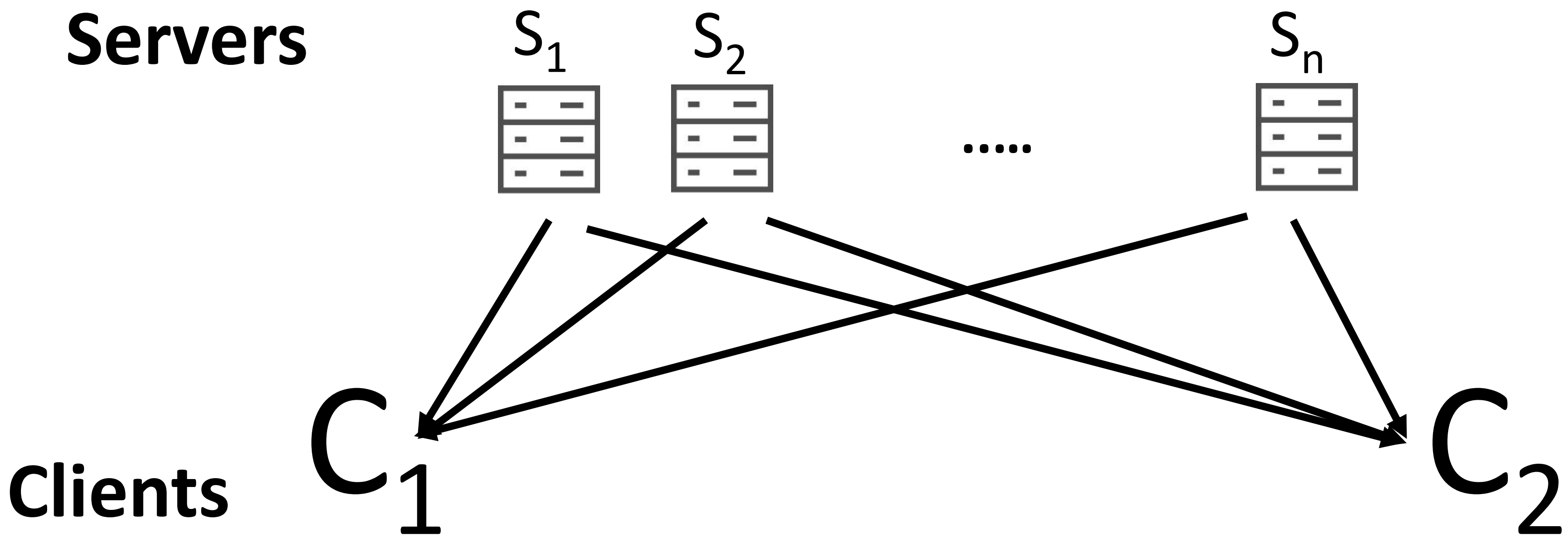
The [IPS08] Compiler – Outer Protocol



The [IPS08] Compiler – Outer Protocol



The [IPS08] Compiler – Outer Protocol



The [IPS08] Compiler – Inner Protocol

Implement server's actions

1. Server's view is additively shared across clients
2. Any passive protocol for server's computation
   a) GMW in the OT/OLE-hybrid for Boolean/Arithmetic computation
   b) FHE based secure computation

The [IPS08] Compiler – Combined Protocol

1. **Watchlist Setup**
   - Obtain random subset of PRG seeds using t-out-of-n OT (done twice)
2. **Views of servers** additively shared among clients
3. **Emulate servers actions** via inner protocol

Optimizing the IPS Compiler [LOP11]

- First work to concretely analyze parameters
- Improved watchlist mechanism (i.e. reduced #servers)
- Room to improve
  - Optimize communication of outer protocol
  - Optimize the analysis
  - No implementation

The [IPS08] Compiler – Our Instantiations

Outer Protocol – New Optimized Protocol

- Inspired from [AHIV17]

Inner Protocol – [GMW87]

Our Approach – Improvements the Outer Protocol

- Optimize parameters – new analysis of adaptive security [AHIV17]
- Batch consistency checks (security with abort)

Our Analysis [AHIV17]

Requirements: $deg = t + e + m < n/2$ and **$e < (n\text{-}deg)/3$**

n = #servers, e = #deviations, t = #watchlists,

m = packing factor

Robustness: Probability of affecting correctness

Prob. deviations are not caught= $(1-e/n)^t$

Prob. bad shares are not caught= $(e+2)/|F|^s + ((2deg+e)/n)^t$

Efficiency: Number of OLEs per mult. = $2(n/m)$

Concrete Parameters

m	e	t	n	n/m
1024	236	469	3922	3.83
2048	301	616	6521	3.18
4096	419	778	11409	2.78
8192	539	1105	20730	2.53
16384	767	1455	38719	2.36
32768	1058	2015	73760	2.25
65536	1458	2831	142513	2.17
131072	2000	4034	278137	2.12
262144	2848	5574	546722	2.08
524288	3959	7928	1080119	2.06

Outer Protocol for Arithmetic 2PC

- Input sharing phase: Additively share all input wires
- For each layer:
    1. Secret share blocks via share packing and send to servers
    2. Servers locally add/multiply values
    3. Return additive shares of output to clients
    4. Degree reduction and rearrange: Apply linear transformations
- After all computation layers
    - Degree test – servers check degree of all input shares
    - Permutation test – servers check all rearrangements
- Reveal outputs

Illustration - Active OLE from Passive OLE

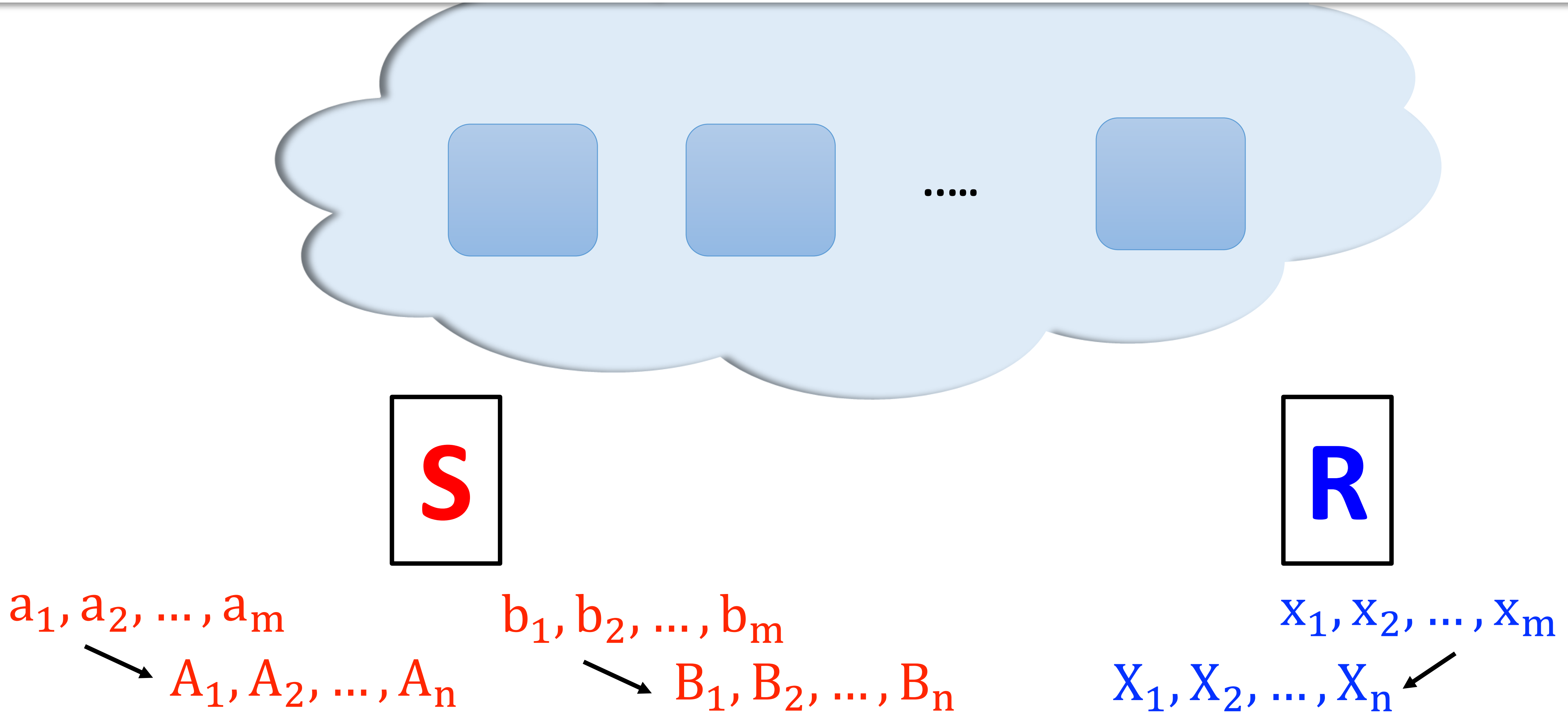
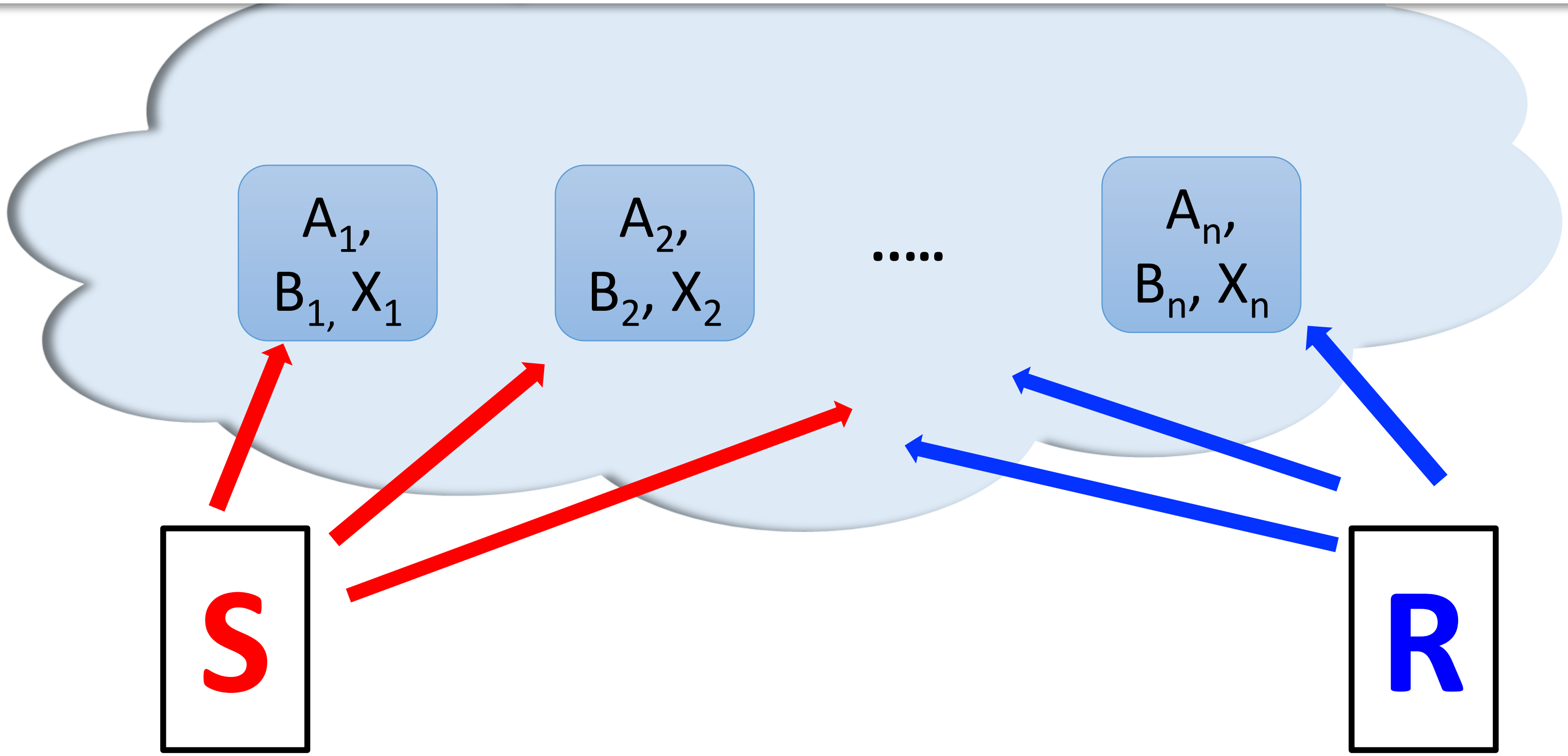


Illustration - Active OLE from Passive OLE

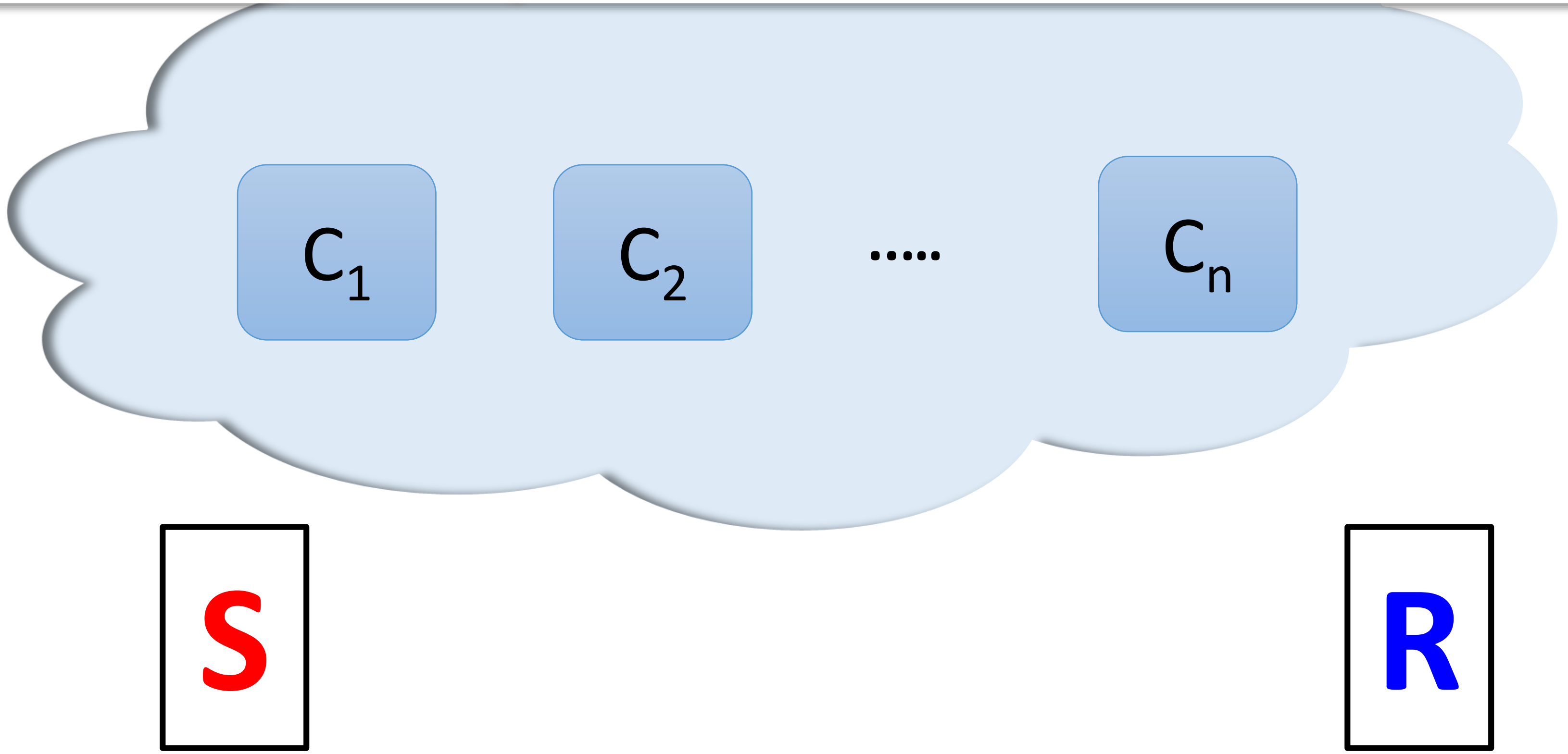


$a_1, a_2, \ldots, a_m \rightarrow A_1, A_2, \ldots, A_n$

$b_1, b_2, \ldots, b_m \rightarrow B_1, B_2, \ldots, B_n$

$x_1, x_2, \ldots, x_m \rightarrow X_1, X_2, \ldots, X_n$

Illustration - Active OLE from Passive OLE



$$C_i = A_i \cdot X_i + B_i$$

$a_1, a_2, \ldots, a_m \rightarrow A_1, A_2, \ldots, A_n$

$b_1, b_2, \ldots, b_m \rightarrow B_1, B_2, \ldots, B_n$

$x_1, x_2, \ldots, x_m \rightarrow X_1, X_2, \ldots, X_n$

Illustration - Active OLE from Passive OLE

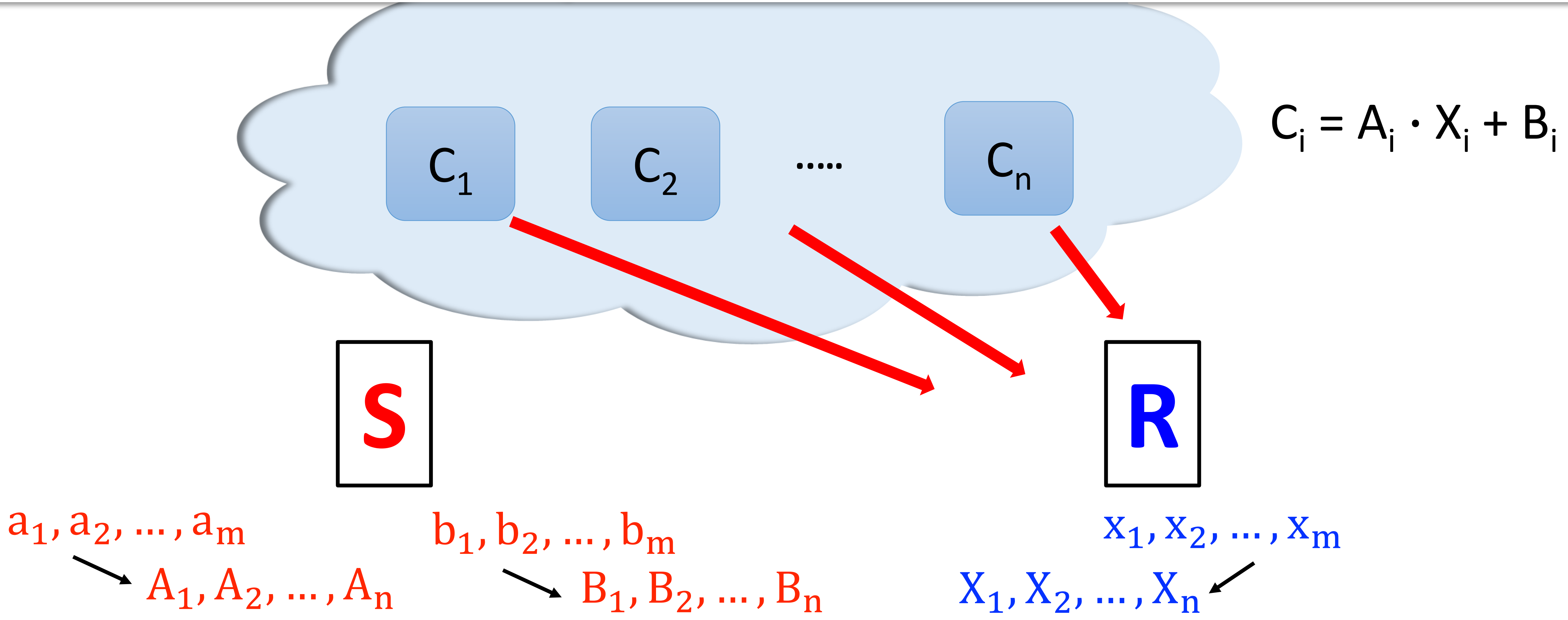
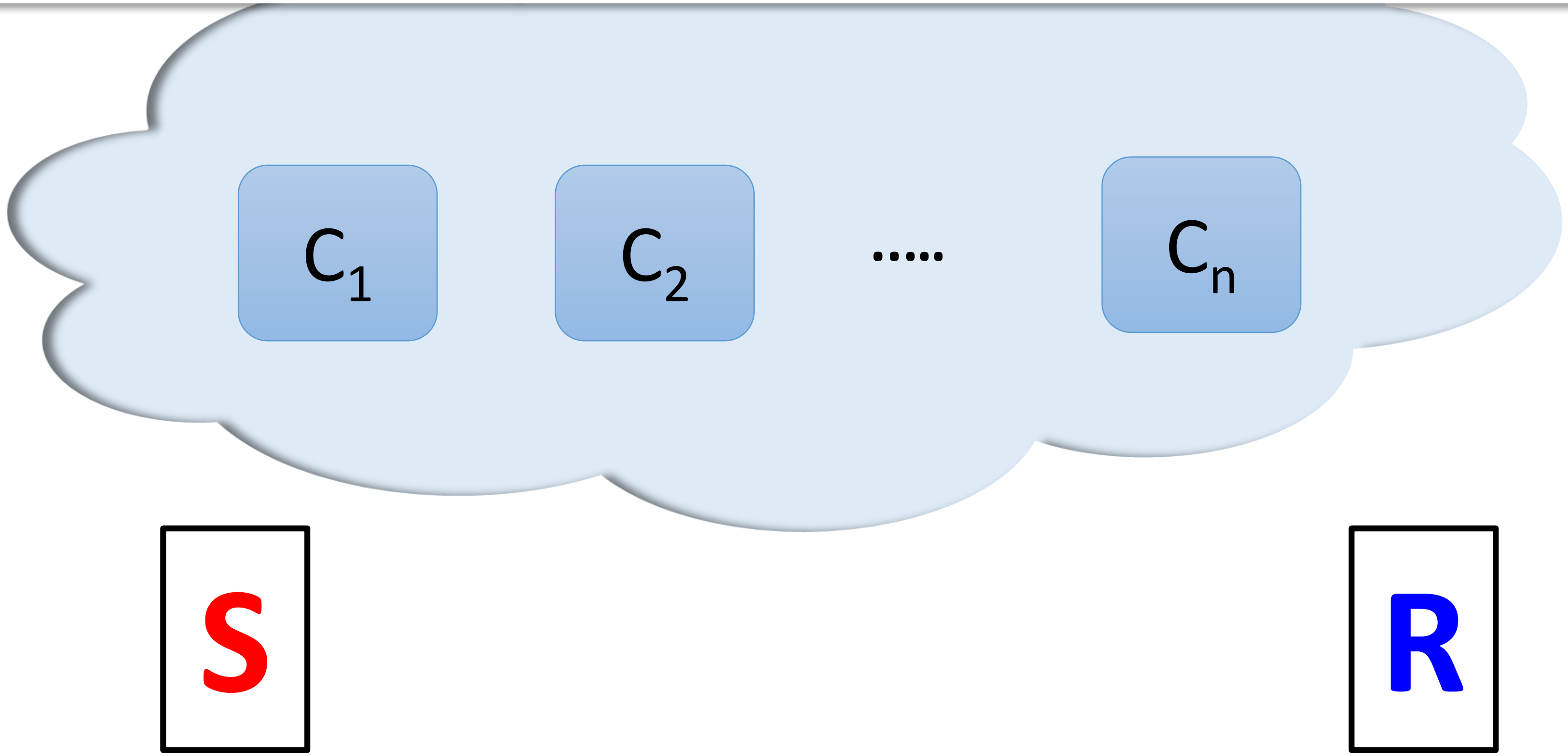


Illustration - Active OLE from Passive OLE



$C_i = A_i \cdot X_i + B_i$

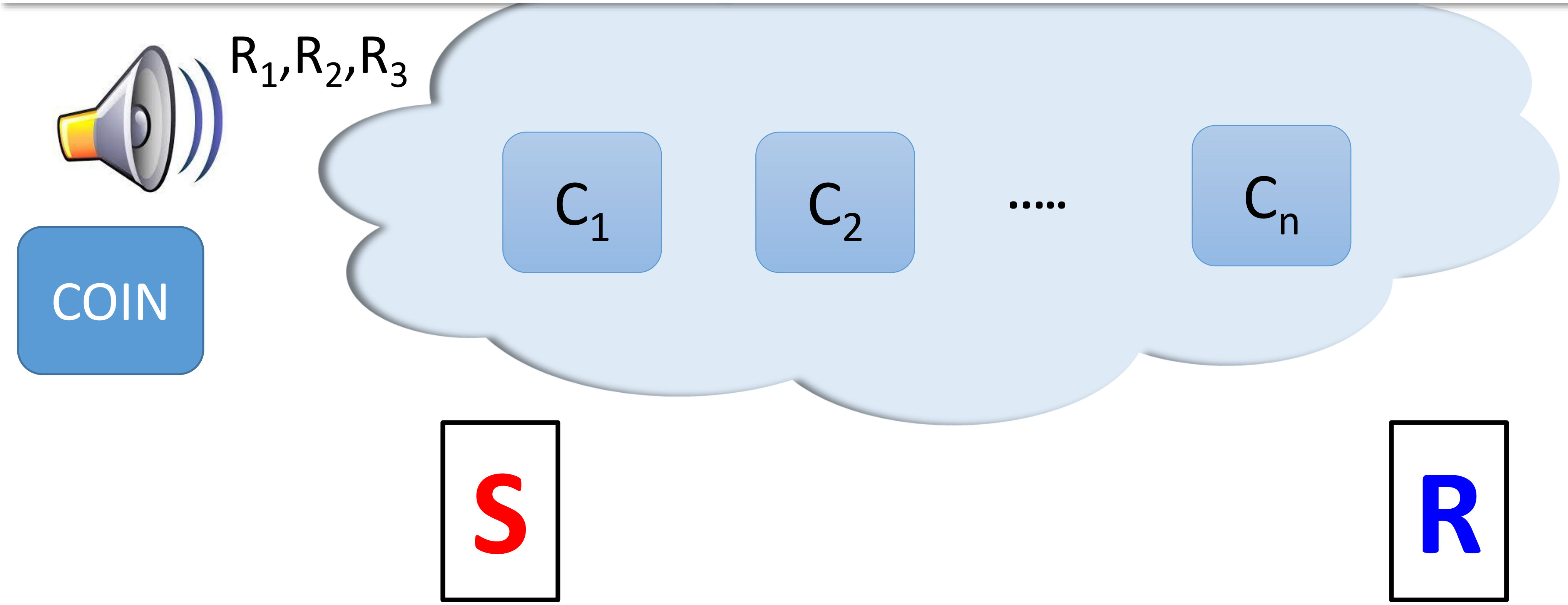
$a_1, a_2, \ldots, a_m \rightarrow A_1, A_2, \ldots, A_n$

$b_1, b_2, \ldots, b_m \rightarrow B_1, B_2, \ldots, B_n$

$x_1, x_2, \ldots, x_m \rightarrow X_1, X_2, \ldots, X_n$

$C_1, C_2, \ldots, C_n \longrightarrow c_1, c_2, \ldots, c_m$

Illustration - Active OLE from Passive OLE



$C_i = A_i \cdot X_i + B_i$

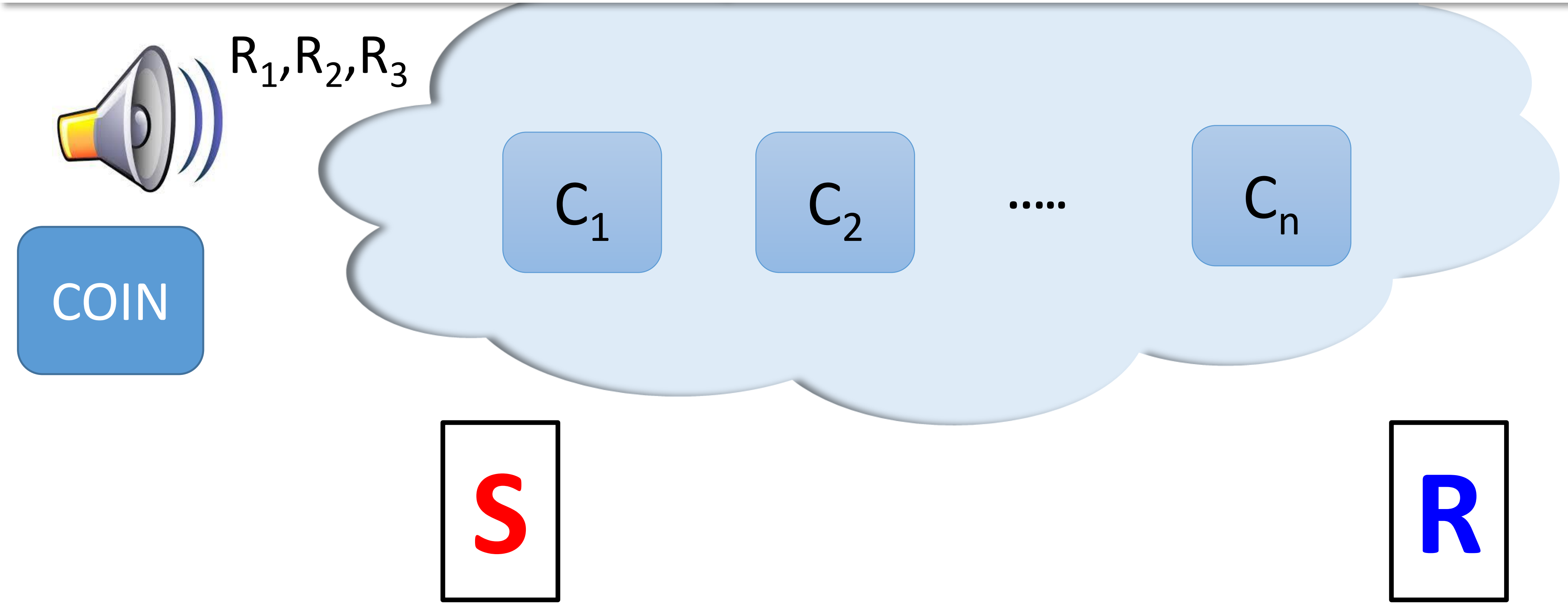
$a_1, a_2, \ldots, a_m \rightarrow A_1, A_2, \ldots, A_n$

$b_1, b_2, \ldots, b_m \rightarrow B_1, B_2, \ldots, B_n$

$x_1, x_2, \ldots, x_m \rightarrow X_1, X_2, \ldots, X_n$

$C_1, C_2, \ldots, C_n \longrightarrow c_1, c_2, \ldots, c_m$

Illustration - Active OLE from Passive OLE



$C_i = A_i \cdot X_i + B_i$

$T_i = R_1 \cdot A_i + R_2 \cdot X_i + R_3 \cdot B_i$

$a_1, a_2, \dots, a_m \rightarrow A_1, A_2, \dots, A_n$

$b_1, b_2, \dots, b_m \rightarrow B_1, B_2, \dots, B_n$

$x_1, x_2, \dots, x_m \rightarrow X_1, X_2, \dots, X_n$

$C_1, C_2, \dots, C_n \longrightarrow c_1, c_2, \dots, c_m$

On Our Computational Complexity

- Recent results achieve constant computation overhead [ADINZ17,BCGGHJ17]

- Our protocol requires log(n) multiplicative overhead
  - Not too bad in practice…

Some Implementation Numbers…

# mults. (x)	Total Time (ms)	Mults. per millisec.	# Field elem.	Comm. per mult. (bits)
1099	78.20	14.21	16384	954
2748	175.40	15.74	32768	763
6280	370.40	17	65536	667
13568	732.00	18.73	131072	618
28672	1338.00	21.56	262144	585
59392	2839.60	21.07	524288	564

Summary

1. First efficient implem. of **general** passive-to-active compiler [ala IPS08]
2. Active OLE that can instantiated from any passive OLE
3. Implementation!
   - Integrating with LWE-based OLE [in progress]

Thank You