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)

sender

a, b ∈ F

OLE

receiver

x ∈ F

ax + b
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

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

3. 2PC based on semi-homomorphihc 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(|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

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

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

   - 22 calls to active OLE
   - 6 log(|F|) calls to active OT
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
   - Requires bit-decomposition

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

   22 calls to active OLE

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

   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

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

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

22 calls to active OLE
6 log(|F|) calls to active OT
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

Real protocol execution
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

Servers: \( S_1 \), \( S_2 \), ..., \( S_n \)

Clients: \( C_1 \), \( C_2 \)
The [IPS08] Compiler – Outer Protocol

Servers

\[ \text{S}_1 \quad \text{S}_2 \quad \ldots \quad \text{S}_n \]

Clients

\[ \text{C}_1 \quad \text{C}_2 \]
The [IPS08] Compiler – Outer 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: \( \text{deg} = t + e + m < n/2 \) and \( e < (n-\text{deg})/3 \)
\( n = \#\text{servers}, \ e = \#\text{deviations}, \ t = \#\text{watchlists}, \ m = \text{packing factor} \)

Robustness: Probability of affecting correctness
Prob. deviations are not caught = \( (1-e/n)^t \)
Prob. bad shares are not caught = \( (e+2)/|F|^s + ((2\text{deg}+e)/n)^t \)

Efficiency: Number of OLEs per mult. = \( 2(n/m) \)
## Concrete Parameters

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

$S$  
$a_1, a_2, ..., a_m$  
$A_1, A_2, ..., A_n$  

$b_1, b_2, ..., b_m$  
$B_1, B_2, ..., B_n$  

$R$  
$x_1, x_2, ..., x_m$  
$X_1, X_2, ..., X_n$
Illustration - Active OLE from Passive OLE

S  ->  A_1, B_1, X_1  ->  A_2, B_2, X_2  ->  ...  ->  A_n, B_n, X_n

S -> a_1, a_2, ..., a_m

R  ->  B_1, B_2, ..., B_n  ->  X_1, X_2, ..., X_m

R  ->  x_1, x_2, ..., x_m
Illustration - Active OLE from Passive OLE

\[ C_i = A_i \cdot X_i + B_i \]
Illustration - Active OLE from Passive OLE

\[ C_i = A_i \cdot X_i + B_i \]
Illustration - Active OLE from Passive OLE

\[ C_i = A_i \cdot X_i + B_i \]

- \( S \): \( a_1, a_2, ..., a_m \rightarrow A_1, A_2, ..., A_n \)
- \( R \): \( x_1, x_2, ..., x_m \rightarrow X_1, X_2, ..., X_n \)
- \( C_1, C_2, ..., C_n \rightarrow c_1, c_2, ..., c_m \)
Illustration - Active OLE from Passive OLE

\[ C_i = A_i \cdot X_i + B_i \]
Illustration - Active OLE from Passive OLE

\[ C_i = A_i \cdot X_i + B_i \]
\[ T_i = R_1 \cdot A_i + R_2 \cdot X_i + R_3 \cdot B_i \]
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...

<table>
<thead>
<tr>
<th># mults. ((x))</th>
<th>Total Time (ms)</th>
<th>Mults. per millisecond</th>
<th># Field elem.</th>
<th>Comm. per mult. (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1099</td>
<td>78.20</td>
<td>14.21</td>
<td>16384</td>
<td>954</td>
</tr>
<tr>
<td>2748</td>
<td>175.40</td>
<td>15.74</td>
<td>32768</td>
<td>763</td>
</tr>
<tr>
<td>6280</td>
<td>370.40</td>
<td>17</td>
<td>65536</td>
<td>667</td>
</tr>
<tr>
<td>13568</td>
<td>732.00</td>
<td>18.73</td>
<td>131072</td>
<td>618</td>
</tr>
<tr>
<td>28672</td>
<td>1338.00</td>
<td>21.56</td>
<td>262144</td>
<td>585</td>
</tr>
<tr>
<td>59392</td>
<td>2839.60</td>
<td>21.07</td>
<td>524288</td>
<td>564</td>
</tr>
</tbody>
</table>
Summary

1. First efficient implem. of \textit{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